DIGITAL RESEARCH®

# CP/M Plus™

(CP/M® Version 3)
Operating System

# User's Guide
# Programmer's Guide
# System Guide

DIGITAL RESEARCH®

# CP/M Plus™

(CP/M® Version 3)
Operating System

# User's Guide
# Programmer's Guide
# System Guide

This manual is reprinted by Commodore Business Machines Inc. with the permission of Digital Research Inc.
P.O. Box 579
160 Central Avenue
Pacific Grove, CA 93950

# CP/M Plus™

(CP/M® Version 3)

Operating System

# User's Guide

## COPYRIGHT



## DISCLAIMER



## NOTICE TO USER

From time to time changes are made in the filenames and in the files actually included on the distribution disk. This manual should not be construed as a representation or warranty that such files or materials and facilities exist on the distribution disk or as part of the materials and programs distributed. Most distribution disks include a "README.DOC" file. This file explains variations from the manual which do constitute modification of the manual and the items included therewith. Be sure to read this file before using the software.

## TRADEMARKS



The *CP/M Plus (CP/M Version 3) Operating System User's Guide* was printed in the United States of America.

First Edition: January 1983
Second Edition: March 1983

# Foreword

Welcome to the world of microcomputers opened to you by your eight-bit microprocessor. Welcome also to the world of application software accessible with your Digital Research CP/M Plus™ operating system, also called CP/M® 3. Digital Research designed CP/M 3 especially for the 8080, 8085, Z80® or equivalent microprocessor that is the heart of your computer.

**What CP/M 3 Does For You**

CP/M 3 manages and supervises your computer's resources, including memory and disk storage, the console (screen and keyboard), printer, and communications devices. It also manages information stored magnetically on disks by grouping this information into files of programs or data. CP/M 3 can copy files from a disk to your computer's memory, or to a peripheral device such as a printer. To do this, CP/M 3 places various programs in memory and executes them in response to commands you enter at your console.

Once in memory, a program executes through a set of steps that instruct your computer to perform a certain task. You can use CP/M 3 to create your own programs, or you can choose from the wide variety of CP/M 3 application programs that entertain you, educate you, and help you solve commercial and scientific problems.

**What You Need to Run CP/M 3 on Your Computer**

Digital Research provides two kinds of CP/M 3 systems: banked and nonbanked. Your computer dealer can tell you if you have a banked or nonbanked system. The banked system requires more memory, but in turn provides more memory space for application programs. The banked version also has additional enhancements that are noted in the text.

The minimum hardware requirement for both versions of CP/M 3 is a computer based on an 8080, 8085, or equivalent microprocessor, a console device (generally a keyboard and display device such as a CRT screen), and at least one floppy disk drive. To use all the capabilities of CP/M 3, you should have two disk drives. At least one should be a single density floppy drive, because CP/M 3 and most CP/M applications are distributed on single density floppy disks.

The nonbanked system requires at least 32K (kilobytes) of Random Access Memory (RAM). The larger banked system requires at least 96K of RAM. If you want to expand beyond these requirements, you will appreciate that the banked system can include up to sixteen banks of memory.

CP/M 3 and its utility programs are distributed on two floppy disks. The system disk contains the operating system and the most commonly used utility programs. A second disk contains additional utilities.

**How To Use CP/M 3 Documentation**

The CP/M 3 documentation set includes three manuals:

- *CP/M Plus (CP/M Version 3) Operating System User's Guide*
- *CP/M Plus (CP/M Version 3) Operating System Programmer's Guide*
- *Programmer's Utilities Guide for the CP/M Family of Operating Systems*

The *CP/M Plus (CP/M Version 3) Operating System User's Guide* introduces you to the CP/M 3 operating system and tells you how to use it. The User's Guide assumes that the version of CP/M 3 on your distribution disk is ready to run on your computer. To use this manual, you must be familiar with the parts of your computer, know how to set it up and turn it on, and how to handle, insert, and store disks. However, you do not need a great deal of experience with computers.

The *CP/M Plus (CP/M Version 3) Operating System Programmer's Guide* presents information for application programmers who are creating or adapting programs to run under CP/M 3. The *Programmer's Utilities Guide for the CP/M Family of Operating Systems* includes information on the CP/M assemblers and debuggers that experienced programmers use to create new CP/M 3 programs.

**How This Guide is Organized**

This guide begins with simple examples, proceeds with basic concepts, then presents a detailed reference section on commands. The first four sections describe CP/M 3 operation for the first-time user. Section 1 introduces CP/M 3 and tells you how to start the operating system, enter commands, edit the command line, and create back-up copies of your distribution disks. Section 2 discusses files, disks, and drives. Section 3 describes how you can use CP/M 3 to manage your printer and console. Section 4 develops the concepts you need to use CP/M 3 commands. If you are new to CP/M, read the first four sections carefully to get a general understanding of how to use CP/M 3 before you proceed to the specific command descriptions.

Section 5 provides detailed information on each CP/M 3 utility program, arranged alphabetically for easy reference. Many of these are programming utilities that you will not use until you start writing your own CP/M 3 programs. Section 6 tells you how to use ED, the CP/M 3 file editor. With ED, you can create and edit program source codes, text, and data files.

Appendix A lists the messages CP/M 3 displays when it encounters special conditions, and describes corrective action where necessary. Appendix B provides an ASCII to hexadecimal conversion table. Appendix C lists the filetypes associated with CP/M 3. Appendix D lists and defines the CP/M 3 control characters. Appendix E provides a glossary of commonly used computer terms.

If you are new to computers, you might find some of the topics, such as the programming utilities, difficult to understand at first. Learning to use your computer is a challenge, and we hope you will find it fun. This book proceeds step-by-step so that you can quickly proceed from opening your new system disk package to mastering CP/M 3's powerful facilities.

# Table of Contents

# Table of Contents (continued)

# Table of Contents (continued)

# Table of Contents (continued)

# Table of Contents (continued)



## Appendixes



## List of Tables

# Table of Contents (continued)



## Figure

# Section 1
# Introduction to CP/M 3

This section tells you how to start CP/M 3, how to enter and edit the command line, and how to make back-up copies of your CP/M 3 distribution disks.

## 1.1 How to Start CP/M 3

Starting or loading CP/M 3 means reading a copy of the operating system from your CP/M 3 system disk (1 of 2 of your distribution disks) into your computer's memory.

First, check that your computer's power is on. Next, insert the CP/M 3 system disk into your initial drive. In this section, assume that the initial drive is A and the disk is removable. Close the drive door. Then, press the RESET or RESTART button. This automatically loads CP/M 3 into memory. This process is called booting, cold starting, or loading the system.

After CP/M 3 is loaded into memory, a message similar to the following is displayed on your screen:

CP/M 3          Version V.V

The version number, represented above by V.V, tells you the version of CP/M 3 that you own. After this display, the following two-character message appears on your screen:

```
A>
```

This is the CP/M 3 system prompt. The system prompt tells you that CP/M 3 is ready to read a command from your keyboard. In this example, the prompt also tells you that drive A is your default drive. This means that until you tell CP/M 3 to do otherwise, it looks for program and data files on the disk in drive A. It also tells you that you are logged in as user 0, by the absence of a user number other than 0.

## 1.2 The Command Line

CP/M 3 performs tasks according to specific commands that you type at your keyboard. A CP/M 3 command line is composed of a command keyword, an optional command tail, and a carriage return keystroke. The command keyword identifies a command (program) to be executed. The command tail can contain extra information for the command, such as a filename or parameters. To end the command line, you must press the carriage return or ENTER key. The following example shows a command line.

```
A>DIR MYFILE
```

The characters that the user types are slanted to distinguish them from characters that the system displays. In this example, DIR is the command keyword and MYFILE is the command tail. The carriage return keystroke does not appear on the screen or in the example. You must remember to press the carriage return key to send a command line to CP/M 3 for processing. Note that the carriage return key can be marked ENTER, RETURN, CR, or something similar on your keyboard. In this guide, RETURN signifies the carriage return key.

As you type characters at the keyboard, they appear on your screen. The single-character position indicator, called the cursor, moves to the right as you type characters. If you make a typing error, press either the BACKSPACE key (if your keyboard has one) or CTRL-H to move the cursor to the left and correct the error. CTRL is the abbreviation for the Control key. To type a control character, hold down the Control key and press the required letter key. For example, to move the cursor to the left, hold down CTRL and press the H key.

You can type the keyword and command tail in any combination of upper-case and lower-case letters. CP/M 3 treats all letters in the command line as upper-case.

Generally, you type a command line directly after the system prompt. However, CP/M 3 does allow spaces between the prompt and the command keyword.

CP/M 3 recognizes two different types of commands: built-in commands and transient utility commands. Built-in commands execute programs that reside in memory as a part of the CP/M 3 operating system. Built-in commands can be executed immediately. Transient utility commands are stored on disk as program files. They must be loaded from disk to perform their task. You can recognize transient utility program files when a directory is displayed on the screen because their filenames are followed by COM. Section 4 presents lists of the CP/M 3 built-in and transient utility commands.

For transient utilities, CP/M 3 checks only the command keyword. If you include a command tail, CP/M 3 passes it to the utility without checking it because many utilities require unique command tails. A command tail cannot contain more than 128 characters. Of course, CP/M 3 cannot read either the command keyword or the command tail until you press the RETURN key.

Let's use one command to demonstrate how CP/M 3 reads command lines. The DIR command, which is an abbreviation for directory, tells CP/M 3 to display a directory of disk files on your screen. Type the DIR keyword after the system prompt, omit the command tail, and press RETURN.

```
A>DIR
```

CP/M 3 responds to this command by writing the names of all the files that are stored on the disk in drive A. For example, if you have your CP/M 3 system disk in drive A, these filenames, among others, appear on your screen:

```
COPYSYS  COM
PIP      COM
SET      COM
```

CP/M 3 recognizes only correctly spelled command keywords. If you make a typing error and press RETURN before correcting your mistake, CP/M 3 echoes the command line followed with a question mark. If you mistype the DIR command, as in the following example, CP/M 3 responds

```
A>DJR
DJR?
```

to tell you that it cannot find the command keyword. To correct simple typing errors, use the BACKSPACE key, or hold down the CTRL key and press H to move the cursor to the left. CP/M 3 supports other control characters that help you efficiently edit command lines. Section 3 tells how to use control characters to edit command lines and other information you enter at your console.

DIR accepts a filename as a command tail. You can use DIR with a filename to see if a specific file is on the disk. For example, to check that the transient utility program COPYSYS.COM is on your system disk, type

```
A>DIR COPYSYS.COM
```

CP/M 3 performs this task by displaying either the name of the file you specified, or the message

```
No File.
```

Be sure you type at least one space after DIR to separate the command keyword from the command tail. If you do not, CP/M 3 responds as follows.

```
A>DIRCOPYSYS.COM
DIRCOPYSYS.COM?
```

## 1.3 Why You Should Back Up Your Files

Humans have faults, and so do computers. Human or computer errors sometimes destroy valuable programs or data files. By mistyping a command, for example, you could accidentally erase a program that you just created or a data file that has been months in the making. A similar disaster could result from an electronic component failure.

Data processing professionals avoid losing programs and data by making copies of valuable files. Always make a working copy of any new program that you purchase and save the original. If the program is accidentally erased from the working copy, you can easily restore it from the original.

It is also wise to make frequent copies of new programs or data files as you develop them. The frequency of making copies varies with each programmer. However, as a general rule, make a copy at the point where it takes ten to twenty times longer to reenter the information than it takes to make the copy.

So far, we have not discussed any commands that change information recorded on your CP/M 3 system disk. Before we do, make a few working copies of the your distribution disks.

## 1.4 How to Make Copies of Your CP/M 3 Disks

To back up your CP/M 3 disks, you need two or more floppy disks for the back-ups. The back-up disks can be new or used. You might want to format new, or reformat used disks with the disk formatting program that accompanies your particular computer. If the disks are used, be sure that there are no other files on the disks.

If your computer's manufacturer has provided a special program to copy disks, you might use it to make back-ups of your distribution disks. Otherwise, use the COPYSYS and PIP utility programs found on your CP/M 3 distribution disks. PIP can copy all program and data files, but only COPYSYS can copy the operating system. Note that the COPYSYS utility distributed by Digital Resarch only functions with eight-inch, single-density drives. However, your computer's manufacturer might have modified COPYSYS to work with your equipment.

This section shows how to make distribution disk back-ups on a system that has two drives: drive A and drive B. Your drives might be named with other letters from the range A through P. To make a copy of your CP/M 3 distribution system disk, labeled 1 of 2, first use the COPYSYS utility to copy the operating system loader. Make sure that your distribution system disk is in drive A, the default drive, and the blank disk is in drive B. Then enter the following command at the system prompt:

```
A>COPYSYS
```

CP/M 3 loads COPYSYS into memory and runs it. COPYSYS displays the following output on your screen. When the program prompts you, press RETURN only when you have verified that the correct disk is in the correct drive.

```
Copysys Ver 3.0

Source drive name (or return for default)  ?A

Source on A then type return

Function complete

Destination drive name (or return to reboot) ?B

Destination on B then type return

Function complete

Do you wish to copy CPM3.SYS?yes
```

(CP/M 3 repeats the above prompts to copy CPM3.SYS.)

```
A>
```

You now have a copy of the operating system only. To copy the remaining files from disk 1 of 2, enter the following PIP command.

```
A>PIP B:=A:*.*
```

This PIP command copies all the files in your disk directory to drive B from drive A. PIP displays the message COPYING followed by each filename as the copy operation proceeds. When PIP finishes copying, CP/M 3 displays the system prompt.

Now you have an exact copy of the distribution disk 1 of 2 in drive B. Remove the original from drive A and store it in a safe place. If your original remains safe and unchanged, you can easily restore your CP/M 3 program files if something happens to your working copy.

Remove the copy from drive B and insert it in drive A. Use this copy as your CP/M 3 system disk to make more back-ups, to try the examples shown throughout this manual, and to start CP/M 3 the next time you turn on your computer. Cold start the computer to check copy operations.

You still need to make a back-up copy of distribution disk 2 of 2. This disk contains programmer's utility programs and source files. Place another new or reformatted disk in drive B. This time, type only the command keyword:

```
A>PIP
```

PIP responds with an asterisk prompt, *. You can now remove disk 1 of 2 from drive A and insert the disk you want to copy, disk 2 of 2. Type the following PIP command after the asterisk prompt, for example,

```
*B:=A:*.*
```

Again, PIP displays the message COPYING, followed by each filename. When PIP completes the copy and displays the asterisk prompt, press RETURN. CP/M 3 then displays the familiar A> system prompt. You now have a copy of disk 2 of 2 in drive B. Remove both 2 of 2 disks and store them in a safe place. You can now reinsert your working system disk and continue to use the system.

*End of Section 1*

# Section 2
# Files, Disks, and Drives

CP/M 3's most important task is to access and maintain files on your disks. With CP/M 3 you can create, read, write, copy, and erase disk files. This section tells you what a file is, how to create, name, and access a file, and how files are stored on your disks. It also tells how to change disks and change the default drive.

## 2.1 What is a File?

A CP/M 3 file is a collection of related information stored on a disk. Every file must have a unique name because CP/M 3 uses that name to access that file. A directory is also stored on each disk. The directory contains a list of the filenames stored on that disk and the locations of each file on the disk.

In general, there are two kinds of files: program (command) files and data files. A program file contains an executable program, a series of instructions that the computer follows step-by-step. A data file is usually a collection of information: a list of names and addresses, the inventory of a store, the accounting records of a business, the text of a document, or similar related information. For example, your computer cannot execute names and addresses, but it can execute a program that prints names and addresses on mailing labels.

A data file can also contain the source code for a program. Generally, a program source file must be processed by an assembler or compiler before it becomes a program file. In most cases, an executing program processes a data file. However, there are times when an executing program processes a program file. For example, the copy program PIP can copy one or more program files.

## 2.2 How Are Files Created?

There are many ways to create a file. One way is to use a text editor. The CP/M 3 text editor ED (described in Section 6) can create a file and assign it the name you specify. You can also create a file by copying an existing file to a new location, perhaps renaming it in the process. Under CP/M 3, you can use the PIP command to copy and rename files. Finally, some programs such as MAC™ create output files as they process input files.

## 2.3 How Are Files Named?

CP/M 3 identifies every file by its unique file specification. A file specification can be simply a one- to eight-character filename, such as:

```
MYFILE
```

A file specification can have four parts: a drive specifier, a filename, a filetype, and a password.

The drive specifier is a single letter (A-P) followed by a colon. Each drive in your system is assigned a letter. When you include a drive specifier as part of the file specification, you are telling CP/M 3 that the file is stored on the disk currently in that drive. For example, if you enter

```
B:MYFILE
```

CP/M 3 looks in drive B for the file MYFILE.

The filename can be from one to eight characters. When you make up a filename, try to let the name tell you something about what the file contains. For example, if you have a list of customer names for your business, you could name the file:

```
CUSTOMER
```

so that the name gives you some idea of what is in the file.

As you begin to use your computer with CP/M 3, you will find that files fall naturally into categories. To help you identify files belonging to the same category, CP/M 3 allows you to add an optional one- to three-character extension, called a filetype, to the filename. When you add a filetype to the filename, separate the filetype from the filename with a period. Try to use three letters that tell something about the file's category. For example, you could add the following filetype to the file that contains a list of customer names:

```
CUSTOMER.NAM
```

When CP/M 3 displays file specifications in response to a DIR command, it adds blanks to short filenames so that you can compare filetypes quickly. The program files that CP/M 3 loads into memory from a disk have different filenames, but all have the filetype COM.

In banked CP/M 3, you can add a password as an optional part of the file specification. The password can be from one to eight characters. If you include a password, separate it from the filetype (or filename, if no filetype is included) with a semicolon, as follows:

```
CUSTOMER.NAM;ACCOUNT
```

If a file has been protected with a password, you must ENTER the password as part of the file specification to access the file. Section 2.7.3 describes passwords in more detail.

We recommend that you create filenames, filetypes, and passwords from letters and numbers. You must not use the following characters in filenames, filetypes, or passwords because they have special meanings for CP/M 3:

< > = , ! | * ? & / $ [ ] ( ) . : ; \ + -

A complete file specification containing all possible elements consists of a drive specification, a primary filename, a filetype, and a password, all separated by their appropriate delimiters, as in the following example:

```
A:DOCUMENT.LAW;SUSAN
```

## 2.4 Do You Have the Correct Drive?

When you type a file specification in a command tail without a drive specifier, the program looks for the file in the drive named by the system prompt, called the default drive. For example, if you type the command

```
A>DIR COPYSYS.COM
```

DIR looks in the directory of the disk in drive A for COPYSYS.COM. If you have another drive, B for example, you need a way to tell CP/M 3 to access the disk in drive B instead. For this reason, CP/M 3 lets you precede a filename with a drive specifier. For example, in response to the command

```
A>DIR B:MYFILE.LIB
```

CP/M 3 looks for the file MYFILE.LIB in the directory of the disk in drive B. When you give a command to CP/M 3, note which disk is in the default drive. Many application programs require that the data files they access be stored in the default drive.

You can also precede a program filename with a drive specifier, even if you use the program filename as a command keyword. For example, if you type the following command

```
A>B:PIP
```

CP/M 3 looks in the directory of the disk in drive B for the file PIP.COM. If CP/M 3 finds PIP on drive B, it loads PIP into memory and executes it.

If you need to access many files on the same drive, you might find it convenient to change the default drive so that you do not need to repeatedly enter a drive specifier. To change the default drive, enter the drive specifier next to the system prompt and press RETURN. In response, CP/M 3 changes the system prompt to display the new default drive:

```
A>B:
B>
```

Unlike the filename and filetype which are stored in the disk directory, the drive specifier for a file changes as you move the disk from one drive to another. Therefore, a file has a different file specification when you move a disk from one drive to another. Section 4 presents more information on how CP/M 3 locates program and data files.

## 2.5 Do You Have the Correct User Number?

CP/M 3 further identifies all files by assigning each one a user number which ranges from 0 to 15. CP/M 3 assigns the user number to a file when the file is created. User numbers allow you to separate your files into sixteen file groups. User numbers are particularly useful for organizing files on a hard disk.

When you use a CP/M 3 utility to create a file, the file is assigned to the current user number, unless you use PIP to copy the file to another user number. You can determine the current user number by looking at the system prompt.

```
4A>         User number 4, drive A
A>          User number 0, drive A
2B>         User number 2, drive B
```

The user number always precedes the drive identifier. User 0, however, is the default user number and is not displayed in the prompt.

You can use the built-in command USER to change the current user number.

```
A>USER 3
3A>
```

You can change both the user number and the drive by entering the new user number and drive specifier together at the system prompt:

```
A>3B:
3B>
```

Most commands can access only those files that have the current user number. For example, if the current user number is 7, a DIR command with no options displays only the files that were created under user number 7. However, if a file resides in user 0 and is marked with a special file attribute, the file can be accessed from any user number. (Section 2.7.1 discusses file attributes.)

## 2.6 Accessing More Than One File

Certain CP/M 3 built-in and transient utilities can select and process several files when special wildcard characters are included in the filename or filetype. A file specification containing wildcards is called an ambiguous filespec and can refer to more than one file because it gives CP/M 3 a pattern to match. CP/M 3 searches the disk directory and selects any file whose filename or filetype matches the pattern.

The two wildcard characters are ?, which matches any single letter in the same position, and *, which matches any character at that position and any other characters remaining in the filename or filetype. The following list presents the rules for using wildcards.

- A ? matches any character in a name, including a space character.
- An * must be the last, or only, character in the filename or filetype. CP/M 3 internally replaces an * with ? characters to the end of the filename or filetype.
- When the filename to match is shorter than eight characters, CP/M 3 treats the name as if it ends with spaces.
- When the filetype to match is shorter than three characters, CP/M 3 treats the filetype as if it ends with spaces.

Suppose, for example, you have a disk that contains the following six files:

`A.COM   AA.COM   AAA.COM   B.COM   A.ASM` and `B.ASM`

The following wildcard specifications match all, or a portion of, these files:

| | |
|---|---|
| `*.*` | is treated as ????????.??? |
| `????????.???` | matches all six names |
| `*.COM` | is treated as ????????.COM |
| `????????.COM` | matches the first four names |
| `?.COM` | matches A.COM and B.COM |
| `?.*` | is treated as ?.??? |
| `?.???` | matches A.COM, B.COM, A.ASM, and B.ASM |
| `A?.COM` | matches A.COM and AA.COM |
| `A*.COM` | is treated as A???????.COM |
| `A???????.COM` | matches A.COM, AA.COM, and AAA.COM |

Remember that CP/M 3 uses wildcard patterns only while searching a disk directory, and therefore wildcards are valid only in filenames and filetypes. You cannot use a wildcard character in a drive specifier. You also cannot use a wildcard character as part of a filename or filetype when you create a file.

## 2.7 How to Protect Your Files

Under CP/M 3 you can organize your files into groups to protect them from accidental change and from unauthorized access. You can specify how your files are displayed in response to a DIR command, and monitor when your files were last accessed or modified. CP/M 3 supports these features by assigning the following to files:

- user numbers
- attributes
- time and date stamps
- passwords (banked CP/M 3 only)

All of this information for each file is recorded in the disk directory.

### 2.7.1 File Attributes

File attributes control how a file can be accessed. When you create a file, CP/M 3 gives it two attributes. You can change the attributes with a SET command.

The first attribute can be set to either DIR (Directory) or SYS (System). This attribute controls whether CP/M 3 displays the file's name in response to a DIR command or DIRSYS command. When you create a file, CP/M 3 automatically sets this attribute to DIR. You can display the name of a file marked with the DIR attribute with a DIR command. If you give a file the SYS attribute, you must use a DIRSYS command to display the filename. Simple DIR and DIRSYS commands display only the filenames created under the current user number.

A file with the SYS attribute has a special advantage when it is created under user 0. When you give a file with user number 0 the SYS attribute, you can read and execute that file from any user number. This feature gives you a convenient way to make your commonly used programs available under any user number. Note, however, that a user 0 SYS file does not appear in response to a DIRSYS command unless 0 is the current user number.

The second file attribute can be set to either R/W (Read-Write) or R/O (Read-Only). If a file is marked R/O, any attempt to write data to that file produces a Read-Only error message. Therefore, you can use the R/O attribute to protect important files. A file with the R/W attribute can be read or written to, or erased at any time, unless the disk is physically write-protected.

### 2.7.2 Date and Time Stamping

If you use date and time stamps, you can quickly locate the most recent copy of a file, and check when it was last updated or changed. You can choose to have the system tell you either when you created the file, or when you last read from or wrote to the file. You use the SET command to enable date and time stamping, and the DIR command with the DATE option to display a file's time and date stamp.

A SET command enables the option you want to monitor. You can use the following commands to enable time and date stamping on a disk, but you must choose between ACCESS and CREATE. If you choose ACCESS, the stamp records the last time the file was accessed. If you choose CREATE, the stamp records when the file was created.

```
A>SET [ACCESS=ON]
A>SET [CREATE=ON]
A>SET [UPDATE=ON]
```

Files created on or copied to a disk that has time and date stamping are automatically stamped. The DATE command allows you to display and reset the time and date that CP/M 3 is using. For a complete discussion of time and date stamping, see the descriptions of the SET and INITDIR commands in Section 5.

### 2.7.3 Passwords (Banked CP/M 3 Only)

Passwords allow you to protect your files from access by other users. You can use passwords to limit access to certain files for security purposes.

The SET utility allows you to enable password protection on a drive, assign a password to SET itself (so that unauthorized users cannot disable password protection on a drive), and assign passwords to specific files that have already been created. You can assign passwords to all program and data files. This means that a command line could require the entry of two passwords in order to execute: one password to access the command program, and a second password to access the file specified in the command tail.

Some CP/M 3 commands and most word processing, accounting, and other application programs running under CP/M 3 do not accept passwords in the command tail. If you want to protect your file and still use those programs, you can set a default password before executing the application program. See the description of the SET command in Section 5 for an explanation of this process.

## 2.8 How Are Files Stored on a Disk?

CP/M 3 records the filename, filetype, password, user number, and attributes of each file in a special area of the disk called the directory. In the directory, CP/M 3 also records which parts of the disk belong to which file.

CP/M 3 allocates directory and storage space for a file as records are added to the file. When you erase a file, CP/M 3 reclaims storage in two ways: it makes the file's directory space available to catalog a different file, and frees the file's storage space for later use. It is this dynamic allocation feature that makes CP/M 3 powerful. You do not have to tell CP/M 3 how big your file will become, because it automatically allocates more storage for a file as needed, and releases the storage for reallocation when the file is erased. Use the SHOW command to determine how much space remains on the disk.

## 2.9 Changing Floppy Disks

CP/M 3 cannot, of course, do anything to a file unless the disk that holds the file is inserted into a drive and the drive is ready. When a disk is in a drive, it is online and CP/M 3 can access its directory and files.

At some time, you will need to take a disk out of a drive and insert another that contains different files. You can replace an online disk whenever you see the system prompt at your console. This is a clear indication that no program is reading or writing to the drive.

You can also remove a disk and insert a new one when an application program prompts you to do so. This can occur, for example, when the data that the program uses does not fit on one floppy disk.

**Note:** you must never remove a disk if a program is reading or writing to it.

You can change disks on the drive without sending any special signals to CP/M 3. This allows you to insert another disk at a program's request and read files from or create files on the new disk.

## 2.10 Protecting a Drive

Under CP/M 3, drives can be marked R/O just as files can be given the R/O attribute. The default state of a drive is R/W. You can give a drive the R/O attribute by using the SET command described in Section 5. To return the drive to R/W, use the SET command or press a CTRL-C at the system prompt.

*End of Section 2*

# Section 3
# Console and Printer

This section describes how CP/M 3 communicates with your console and printer. It tells how to start and stop console and printer output, edit commands you enter at your console, and redirect console and printer input and output. It also explains the concept of logical devices under CP/M 3.

## 3.1 Controlling Console Output

Sometimes CP/M 3 displays information on your screen too quickly for you to read it. Sometimes an especially long display scrolls off the top of your screen before you have a chance to study it. To ask the system to wait while you read the display, hold down the CONTROL (CTRL) key and press S. A CTRL-S keystroke causes the display to pause. When you are ready, press CTRL-Q to resume the display. If you press any key besides CTRL-Q during a display pause, CP/M 3 sounds the console bell or beeper.

DIR, TYPE, and other CP/M 3 utilities support automatic paging at the console. This means that if the program's output is longer than what the screen can display at one time, the display automatically halts when the screen is filled. When this occurs, CP/M 3 prompts you to press RETURN to continue.

## 3.2 Controlling Printer Output

You can also use a control command to echo console output to the printer. To start printer echo, press a CTRL-P. To stop, press CTRL-P again. While printer echo is in effect, any characters that appear on your screen are listed at your printer.

You can use printer echo with a DIR command to make a list of files stored on a floppy disk. You can also use CTRL-P with CTRL-S and CTRL-Q to make a hard copy of part of a file. Use a TYPE command to start a display of the file at the console. When the display reaches the part you need to print, press CTRL-S to stop the display, CTRL-P to enable printer echo, and then CTRL-Q to resume the display and start printing. You can use another CTRL-S, CTRL-P, CTRL-Q sequence to terminate printer echo.

## 3.3 Console Line Editing

You can correct simple typing errors with the BACKSPACE key. CP/M 3 also supports additional line-editing functions for banked and nonbanked systems that you perform with control characters. You can use the control characters to edit command lines or input lines to most programs.

### 3.3.1 Line Editing in Nonbanked CP/M 3

Nonbanked CP/M 3 allows you to edit your command line using the set of characters listed in Table 3-1. To edit a command line in nonbanked CP/M 3, use control characters to delete characters left of the cursor, then replace them with new characters.

In the following example command line, the command keyword PIP is mistyped. The underbar represents the cursor.

```
A>POP A:=B:*.*_
```

To move the cursor to the letter O, hold down the CTRL key and press the letter H eleven times. CTRL-H deletes characters as it moves the cursor left, leaving the following command line:

```
A>P_
```

Now, type the correct letters, press RETURN, and send the command to CP/M 3.

```
A>PIP A:=B:*.*_
```

Table 3-1 lists all line-editing control characters for nonbanked CP/M 3.

**Table 3-1. Nonbanked CP/M 3 Line-editing Control Characters**

| *Character* | *Meaning* |
|---|---|
| CTRL-E | Forces a physical carriage return but does not send the command line to CP/M 3. Moves the cursor to the beginning of the next line without erasing your previous input. |
| CTRL-H | Deletes a character and moves the cursor left one character position. |
| CTRL-I | Moves the cursor to the next tab stop. Tab stops are automatically set at each eighth column. Has the same effect as pressing the TAB key. |
| CTRL-J | Sends the command line to CP/M 3 and returns the cursor to the left of the current line. Has the same effect as a RETURN or a CTRL-M. |
| CTRL-M | Sends the command line to CP/M 3 and returns the cursor to the left of the current line. Has the same effect as a RETURN or a CTRL-J. |
| CTRL-R | Places a # at the current cursor location, moves the cursor to the next line, and displays any partial command you typed so far. |
| CTRL-U | Discards all the characters in the command line, places a # at the current cursor position, and moves the cursor to the next command line. |
| CTRL-X | Discards all the characters in the command line, and moves the cursor to the beginning of the current line. |

You probably noticed that some control characters have the same meaning. For example, the CTRL-J and CTRL-M keystrokes have the same effect as pressing the RETURN key; all three send the command line to CP/M 3 for processing. Also, CTRL-H has the same effect as pressing the BACKSPACE key.

### 3.3.2 Line Editing in Banked CP/M 3

Banked CP/M 3 allows you to edit your command line without deleting all characters. Using the line-editing control characters listed in Table 3-2, you can move the cursor left and right to insert and delete characters in the middle of a command line. You do not have to retype everything to the right of your correction. In banked CP/M 3, you can press RETURN when the cursor is in any position in the command line; CP/M 3 reads the entire command line. You can also recall a command for reediting and reexecution.

In the following sample session, the user mistyped PIP, and CP/M 3 returned an error message. The user recalls the erroneous command line by pressing CTRL-W and corrects the error (the underbar represents the cursor):

```
A>POP A:=B:*.*_               (PIP mistyped)
POP?

A>POP A:=B:*.*_               (CTRL-W recalls the line)

A>POP A:=B:*.*                (CTRL-B to beginning of line)

A>POP A:=B:*.*                (CTRL-F to move cursor right)

A>PP A:=B:*.*                 (CTRL-G to delete error)

A>PIP A:=B:*.*                (type I to correct the command name)
```

To execute the corrected command line, the user can press return even though the cursor is in the middle of the line. A return keystroke, or one of its equivalent control characters, not only executes the command, but also stores the command in a buffer so that you can recall it for editing or reexecution by pressing CTRL-W.

When you insert a character in the middle of a line, characters to the right of the cursor move to the right. If the line becomes longer than your screen is wide, characters disappear off the right side of the screen. These characters are not lost. They reappear if you delete characters from the line or if you press CTRL-E when the cursor is in the middle of the line. CTRL-E moves all characters to the right of the cursor to the next line on the screen.

Table 3-2 gives a complete list of line-editing control characters for a banked CP/M 3 system.

**Table 3-2. Banked CP/M 3 Line-editing Control Characters**

| *Character* | *Meaning* |
|---|---|
| CTRL-A | Moves the cursor one character to the left. |
| CTRL-B | Moves the cursor to the beginning of the command line without having any effect on the contents of the line. If the cursor is at the beginning, CTRL-B moves it to the end of the line. |
| CTRL-E | Forces a physical carriage return but does not send the command line to CP/M 3. Moves the cursor to the beginning of the next line without erasing the previous input. |
| CTRL-F | Moves the cursor one character to the right. |
| CTRL-G | Deletes the character indicated by the cursor. The cursor does not move. |
| CTRL-H | Deletes a character and moves the cursor left one character position. |
| CTRL-I | Moves the cursor to the next tab stop. Tab stops are automatically set at each eighth column. Has the same effect as pressing the TAB key. |
| CTRL-J | Sends the command line to CP/M 3 and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or a CTRL-M keystroke. |
| CTRL-K | Deletes to the end of the line from the cursor. |
| CTRL-M | Sends the command line to CP/M 3 and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or a CTRL-J keystroke. |
| CTRL-R | Retypes the command line. Places a # at the current cursor location, moves the cursor to the next line, and retypes any partial command you typed so far. |

**Table 3-2. (continued)**

| *Character* | *Meaning* |
|---|---|
| CTRL-U | Discards all the characters in the command line, places a # at the current cursor position, and moves the cursor to the next line. However, you can use a CTRL-W to recall any characters that were to the left of the cursor when you pressed CTRL-U. |
| CTRL-W | Recalls and displays previously entered command line both at the operating system level and within executing programs, if the CTRL-W is the first character entered after the prompt. CTRL-J, CTRL-M, CTRL-U, and RETURN define the command line you can recall. If the command line contains characters, CTRL-W moves the cursor to the end of the command line. If you press RETURN, CP/M 3 executes the recalled command. |
| CTRL-X | Discards all the characters left of the cursor and moves the cursor to the beginning of the current line. CTRL-X saves any characters right of the cursor. |

You probably noticed that some control characters have the same meaning. For example, the CTRL-J and CTRL-M keystrokes have the same effect as pressing the RETURN key; all three send the command line to CP/M 3 for processing. Also, CTRL-H has the same effect as pressing the BACKSPACE key. Notice that when a control character is displayed on your screen, it is preceded by an up-arrow, ↑. For example, a CTRL-C keystroke appears as ↑C on your screen.

## 3.4 Redirecting Input and Output

CP/M 3's PUT command allows you to direct console or printer output to a disk file. You can use a GET command to make CP/M 3 or a utility program take console input from a disk file. The following examples illustrate some of the conveniences GET and PUT offer.

You can use a PUT command to direct console output to a disk file as well as the console. With PUT, you can create a disk file containing a directory of all files on that disk, as follows:

```
A>PUT CONSOLE OUTPUT TO FILE DIR.PRN
Putting console output to file: DIR.PRN

A>DIR
A: FILENAME TEX : FRONT    TEX : FRONT    BAK : ONE      BAK : THREE    TEX
A: FOUR     TEX : ONE      TEX : LINEDIT  TEX : EXAMP1   TXT : TWO      BAK
A: TWO      TEX : THREE    BAK : EXAMP2   TXT

A>TYPE DIR.PRN
A: FILENAME TEX : FRONT    TEX : FRONT    BAK : ONE      BAK : THREE    TEX
A: FOUR     TEX : ONE      TEX : LINEDIT  TEX : EXAMP1   TXT : TWO      BAK
A: TWO      TEX : THREE    BAK : EXAMP2   TXT
```

You can use a similar PUT command to direct printer output to a disk file as well as the printer.

A GET command can direct CP/M 3 or a program to read a disk file for console input instead of the keyboard. If the file is to be read by CP/M 3, it must contain standard CP/M 3 command lines. If the file is to be read by a utility program, it must contain input appropriate for that program. A file can contain both CP/M 3 command lines and program input if it also includes a command to start a program.

You add or omit the SYSTEM option in the GET command line to specify whether CP/M 3 or a utility program is to start reading the file, as shown in the following sample session. If you omit the SYSTEM option, the system prompt returns so that you can initiate the program that is to take input from the specified file. If you include the SYSTEM option, CP/M 3 immediately takes input from the specified file.

```
3A>type pip.dat
b:=front.tex
b:=one.tex
b:=two.tex

3A>get console input from file pip.dat

Getting console input from file: PIP.DAT
3A>pip
CP/M 3 PIP VERSION 3.0
*b:=front.tex

*b:=one.tex

*b:=two.tex

*^<cr>

3A>type ccp.dat
dir
show
dirsys

3A>get console input from file ccp.dat [system]
Getting console input from file: CCP.DAT
3A>dir

A: FILENAME TEX : FRONT    TEX : FRONT    BAK : ONE      BAK : THREE    TEX
A: FOUR     TEX : ONE      TEX : LINEDIT  TEX : EXAMP1   TXT : TWO      BAK
A: TWO      TEX : EXAMP3       : EXAMP2   TXT : PIP      DAT : EXAMP4
A: THREE    BAK : EXAMP5       : CCP      DAT

3A>show
A: RW, Space:     3,392k
B: RW, Space:       452k

3A>dirsys
NON-SYSTEM FILE(S) EXIST
```

See the descriptions of GET and PUT in Section 5 for more ways to use redirected input and output.

## 3.5 Assigning Logical Devices

Most CP/M 3 computer systems have a traditional console with a keyboard and screen display. Many also have letter-quality printers. If you use your computer for unusual tasks, you might want to add a different kind of character device to your system: a line printer, a teletype terminal, a modem, or even a joystick for playing games. To keep track of these physically different input and output devices, CP/M 3 associates different physical devices with logical devices. Table 3-3 gives the names of CP/M 3 logical devices. It also shows the physical devices assigned to these logical devices in the distributed CP/M 3 system.

**Table 3-3. CP/M 3 Logical Devices**

| *Logical Device Name* | *Device Type* | *Physical Device Assignment* |
|---|---|---|
| CONIN: | Console input | Keyboard |
| CONOUT: | Console output | Screen |
| AUXIN: | Auxiliary input | Null |
| AUXOUT: | Auxiliary output | Null |
| LST: | List output | Printer |

In some implementations of CP/M 3, you can change these assignments with a DEVICE command. If your system supports the DEVICE command, you can, for example, assign AUXIN and AUXOUT to a modem so that your computer can communicate with others over the telephone.

*End of Section 3*

# Section 4
# CP/M 3 Command Concepts

As we discussed in Section 1, a CP/M 3 command line consists of a command keyword, an optional command tail, and a carriage return keystroke. This section describes the two kinds of programs the command keyword can identify, and tells how CP/M 3 searches for a program file on a disk. This section also tells how to execute multiple CP/M 3 commands, and how to reset the disk system.

## 4.1 Two Kinds of Commands

A command keyword identifies a program that resides either in memory as part of CP/M 3, or on a disk as a program file. Commands that identify programs in memory are called built-in commands. Commands that identify program files on a disk are called transient utility commands.

CP/M 3 has six built-in commands and over twenty transient utility commands. You can add utilities to your system by purchasing various CP/M 3-compatible application programs. If you are an experienced programmer, you can also write your own utilities that operate with CP/M 3.

## 4.2 Built-in Commands

Built-in commands are part of CP/M 3 and are always available for your use regardless of which disk you have in which drive. Built-in commands reside in memory as a part of CP/M 3 and therefore execute more quickly than the transient utilities.

Some built-in commands have options that require support from a related transient utility. The related transient has the same name as the built-in and has a filetype of COM. This type of transient utility is loaded only when a built-in command line contains options that cannot be performed by the built-in command.

If you include certain options in the command tail for a built-in command, CP/M 3 might return a `.COM Required` message. This means that the command tail options require support from a related transient utility and CP/M 3 could not find that program file. The following files must be accessible to support all the functions these built-ins offer: ERASE.COM, RENAME.COM, TYPE.COM, and DIR.COM.

Section 5 explains in detail the built-in commands listed in Table 4-1.

**Table 4-1. Built-in Commands**

| *Command* | *Function* |
|---|---|
| DIR | Displays filenames of all files in the directory except those marked with the SYS attribute. |
| DIRSYS | Displays filenames of files marked with the SYS (system) attribute in the directory. |
| ERASE | Erases a filename from the disk directory and releases the storage space occupied by the file. |
| RENAME | Renames a disk file. |
| TYPE | Displays contents of an ASCII (TEXT) file at your screen. |
| USER | Changes to a different user number. |

CP/M 3 allows you to abbreviate the built-in commands as follows:

| | |
|---|---|
| DIRSYS | DIRS |
| ERASE | ERA |
| RENAME | REN |
| TYPE | TYP |
| USER | USE |

## 4.3 Transient Utility Commands

When you enter a command keyword that identifies a transient utility, CP/M 3 loads the program file from the disk and passes it any filenames, data, or parameters you entered in the command tail. Section 5 provides the operating details for the CP/M 3 transient utilities listed in Table 4-2.

**Table 4-2. Transient Utility Commands**

| *Name* | *Function* |
|---|---|
| COPYSYS | Creates a new boot disk. |
| DATE | Sets or displays the date and time. |
| DEVICE | Assigns logical CP/M devices to one or more physical devices, changes device driver protocol and baud rates, or sets console screen size. |
| DUMP | Displays a file in ASCII and hexadecimal format. |
| ED | Creates and alters character files. |
| GET | Temporarily gets console input from a disk file rather than the keyboard. |
| HELP | Displays information on how to use CP/M 3 commands. |
| HEXCOM | Uses the output from MAC to produce a program file. |
| INITDIR | Initializes a disk directory to allow time and date stamping. |
| LINK | Links REL (relocatable) program modules produced by RMAC™ (relocatable macro assembler) and produces program files. |
| MAC | Translates assembly language programs into machine code form. |
| PIP | Copies files and combines files. |

Table 4-2. (continued)

| *Name* | *Function* |
|---|---|
| PUT | Temporarily directs printer or console output to a disk file. |
| RMAC | Translates assembly language programs into relocatable program modules. |
| SET | Sets file options including disk labels, file attributes, type of time and date stamping, and password protection. |
| SETDEF | Sets system options including the drive search chain. |
| SHOW | Displays disk and drive statistics. |
| SID | Helps you check your programs and interactively correct programming errors. |
| SUBMIT | Automatically executes multiple commands. |
| XREF | Produces a cross-reference list of variables used in an assembler program. |

## 4.4 How CP/M 3 Searches for Program and Data Files

This section describes how CP/M 3 searches for program and data files on disk. If it appears that CP/M 3 cannot find a program file you specified in a command line, the problem might be that CP/M 3 is not looking on the drive where the file is stored. Therefore, you need to understand the steps CP/M 3 follows in searching for program and data files.

### 4.4.1 Finding Data Files

As you recall, when you enter a command line, CP/M 3 passes the command tail to the program identified by the command keyword. If the command tail contains a file specification, the program calls CP/M 3 to search for the data file. If CP/M 3 cannot find the data file, the program displays an error message at the console. Typically, this message is `File not found` or `No File`, but the message depends on the program identified by the command keyword.

If you do not include a drive specifier with the filename in a command tail, CP/M 3 searches the directory of the current user number on the default drive. If the file is not there, CP/M 3 looks for the file with the SYS attribute in the directory of user 0 on the default drive. If CP/M 3 finds the file under user 0, it allows the program Read-Only access to the file. For example, if you enter the following command line,

```
3A>TYPE MYFILE.TXT
```

CP/M 3 first searches the directory for user 3 on drive A. If it does not find MYFILE.TXT there, it searches the directory of user 0 on drive A for MYFILE.TXT marked with the SYS attribute. If the file is not in either directory, CP/M 3 returns control to TYPE, which then displays `No File`.

Some CP/M 3 utilities, such as PIP and DIR, restrict their file search to the current user number. Because CP/M 3 does not allow Read-Write access to SYS files, ERASE and RENAME also restrict their search to the current user number.

The search procedure is basically the same if you do include a drive specifier with the filename. CP/M 3 first looks in the directory of the current user number on the specified drive. Then, if it does not find the file, it looks in the directory for user 0 on the specified drive for the file with the SYS attribute. If CP/M 3 does not find the data file after these two searches, it displays an error message.

### 4.4.2 Finding Program Files

The search procedure for a program file can be very different from a data file search. This is because you can use the SETDEF command described in Section 5 to define the search procedure you want CP/M 3 to follow when it is looking for a program file. With SETDEF you can ask CP/M 3 to make as many as sixteen searches when you do not include a drive specifier before the command keyword, but that is a rare case! We will begin by describing how CP/M 3 searches for program files when you have not yet entered a SETDEF command.

If a command keyword identifies a transient utility, CP/M 3 looks for that program file on the default or specified drive. It looks under the current user number, and then under user 0 for the same file marked with the SYS attribute. At any point in the search process, CP/M 3 stops the search if it finds the program file. CP/M 3 then loads the program into memory and executes it. When the program terminates, CP/M 3 displays the system prompt and waits for your next command. However, if CP/M 3 does not find the command file, it repeats the command line followed by a question mark, and waits for your next command.

If you include a drive specifier before the command keyword, you are telling CP/M 3 precisely where to look for the program file. Therefore, CP/M 3 searches only two locations: the directory for the current user on the specified drive, and then for user 0 on the specified drive, before it repeats the command line with a question mark. For example, if you enter

```
4C>A:SHOW [SPACE]
```

CP/M 3 looks on drive A, user 4 and then user 0 for the file SHOW.COM.

If you do not include a drive specifier before the command keyword, CP/M 3 searches directories in a sequence called a drive chain. When you first receive CP/M 3, there is only one drive in your chain, the default drive. Unless you change the chain with a SETDEF command, CP/M 3 looks in two places for the program file. For example, if you enter

```
7E>SHOW [SPACE]
```

CP/M 3 searches the following locations for the file SHOW.COM:

1. drive E, user 7
2. drive E, user 0

Remember that a SHOW.COM file under user 0 must be marked with the SYS attribute or else CP/M 3 cannot find it. Use a SET command to give program files under user 0 to the SYS attribute because they can then be accessed automatically from all other user areas. You do not have to duplicate frequently used program files in all user areas on all drives.

When you use a SETDEF command to define your own drive chain, include the default drive, and the drive that contains your most frequently used utilities. For an example, assume you defined your drive chain as * (the default drive) and drive A. When you enter the following command:

```
2D>SHOW [SPACE]
```

CP/M 3 looks for SHOW.COM in the following sequence:

1. drive D, user 2
2. drive D, user 0
3. drive A, user 2
4. drive A, user 0

You can include your default drive in your drive chain with an option in a SETDEF command. Any drive chain you specify with SETDEF remains in effect until you restart or reset the system.

You can also use a SETDEF command to enable automatic submit in your drive chain. See Section 4.5 for a description of automatic submit.

## 4.5 Executing Multiple Commands

In the examples so far, CP/M 3 has executed only one command at a time. CP/M 3 can also execute a sequence of commands. You can enter a sequence of commands at the system prompt, or you can put a frequently needed sequence of commands in a disk file. Once you have stored the sequence in a disk file, you can execute the sequence whenever you need to with a SUBMIT command.

To enter multiple commands at the system prompt, separate each command keyword and associated command tail from the next keyword with an exclamation point, !. When you complete the sequence, press RETURN. CP/M 3 executes your commands in order:

```
3A>dirsys!dir examp*.*!show [space]
NON-SYSTEM FILE(S) EXIST
3A>dir examp*.*
A: EXAMP7      : EXAMP1   TXT : EXAMP3      : EXAMP2   TXT : EXAMP4
A: EXAMP5      : EXAMP6
3A>show [space]

A: RW, Space:     3,344k
```

If you find you need to execute the same command sequence frequently, store the sequence in a disk file. To create this file, use ED or another character file editor. The file must have a filetype of SUB. Each command in the file must start on a new line. For example, an UPDATE.SUB file might look like this:

```
DIR A:*.COM
ERA B:*.COM
PIP B:=A:*.COM
```

To execute this list, enter the following command:

```
A>SUBMIT UPDATE
```

The SUBMIT utility passes each command to CP/M 3 for sequential execution. While SUBMIT executes, the commands are usually echoed at the console, as well as any program's screen display, such as the directory or PIP's "COPYING..." message. When one command completes, the system prompt reappears either with the next command in the SUB file, or, when the SUB file is exhausted, by itself to wait for your next command from the keyboard.

If PROFILE exists, PROFILE.SUB is a special submit file that CP/M 3 automatically executes at each cold start. This feature is especially convenient if you regularly execute a standard set of commands, such as SETDEF and DATE SET, before beginning a work session. A PROFILE.SUB might already exist on your distribution disk. If not, you can create one using ED or another editor.

The description of the SUBMIT utility in Section 5 gives more details on how to create a SUB file and use SUBMIT parameters to pass options to the programs to be executed.

You can also use CTRL-C to reset the disk system. This is sometimes called a warm start. When you press CTRL-C and the cursor is at the system prompt, CP/M 3 logs out all the active drives, then logs in the default drive. The active drives are any drives you have accessed since the last cold or warm start. A SHOW [SPACE] command displays the remaining space on all active drives. In the following example, SHOW [SPACE] indicates that three drives are active. However, if you press CTRL-C immediately after this display and then enter another SHOW [SPACE] command, only the space for the default drive, A, is displayed.

## 4.6 Terminating Programs

You can use the two keystroke command CTRL-C to terminate program execution or reset the disk system. To enter a CTRL-C command, hold down the CTRL key and press C.

Not all application programs that run under CP/M can be terminated by a CTRL-C. However, most of the transient utilities supplied with CP/M 3 can be terminated immediately by a CTRL-C keystroke. If you try to terminate a program while it is sending a display to the screen, you might need to press a CTRL-S to halt the display before entering CTRL-C.

You can also use CTRL-C to reset the disk system. This is sometimes called a warm start. When you press CTRL-C and the cursor is at the system prompt, CP/M 3 logs out all the active drives, then logs in the default drive. The active drives are any drives you have accessed since the last cold or warm start. A SHOW [SPACE] command displays the remaining space on all active drives. In the following example, SHOW [SPACE] indicates that three drives are active. However, if you press CTRL-C immediately after this display and then enter another SHOW [SPACE] command, only the space for the default drive, A, is displayed.

```
A>SHOW [SPACE]
A: RW, Space:      9,488k
B: RO, Space:      2,454k
C: RO, Space:      1,665k
A>^C
A>SHOW [SPACE]
A: RW, Space:      9,488k
```

## 4.7 Getting Help

CP/M 3 includes a transient utility command called HELP that can display a summary of what you need to know to use each command described in this manual. To get help, simply enter the command:

```
A>HELP
```

In response, the HELP utility displays a list of topics for which summaries are available. After HELP lists the topics available, it displays its own prompt:

```
HELP>
```

To this prompt, you can enter one of the topics presented in the list, for example,

```
HELP>SHOW
```

After displaying a summary of the SHOW command, HELP lists subtopics that detail different aspects of the SHOW command. To display the information on a subtopic when you have just finished reading the main topic, enter the name of the subtopic preceded by a period.

```
HELP>.OPTIONS
```

In the preceding example, HELP then displays the options available for the SHOW command. As you become familiar with HELP, you might want to call a HELP subtopic directly from the system prompt as follows:

```
A>HELP SHOW OPTIONS
```

HELP lets you learn the basic CP/M 3 commands quickly. You might find that you reference the command summary in Section 5 only when you need details not provided in the HELP summaries. When you add new utilities, you can modify HELP to add or subtract topics, or even modify the summaries HELP presents. See the description of HELP in Section 5 for complete details.

*End of Section 4*

# Section 5
# Command Summary

This section describes the commands and programs supplied with your CP/M 3 operating system. The commands are in alphabetical order. Each command is followed by a short explanation of its operation and examples. More complicated commands are described later in detail. For example, ED is described in Section 6. Other commands, such as SID and MAC, are described fully in other CP/M manuals.

CP/M 3 has replaced some commands from previous CP/M versions. MAC replaces ASM; SHOW and DIR include the previous STAT functions; and SID replaces DDT.

## 5.1 Let's Get Past the Formalities

This section describes the parts of a file specification in a command line. There are four parts in a file specification; to avoid confusion, each part has a formal name:

- drive specifier—the optional disk drive A, B, C, ...P that contains the file or group of files to which you are referring. If a drive specifier is included in your command line, a colon must follow it.
- filename—the one- to eight-character first name of a file or group of files.
- filetype—the optional one- to three-character category name of a file or group of files. If the filetype is present, a period must separate it from the filename.
- password—the optional one- to eight-character password which allows you to protect your files. It follows the filetype, or the filename if no filetype is assigned, and is preceded by a semicolon.

If you do not include a drive specifier, CP/M 3 automatically uses the default drive. If you omit the period and the filetype, CP/M 3 automatically includes a filetype of three blanks.

This general form is called a file specification. A file specification names a particular file or group of files in the directory of the on-line disk given by the drive specifier. For example,

```
B:MYFILE.DAT
```

is a file specification that indicates drive B:, filename MYFILE, and filetype DAT. File specification is abbreviated to

filespec

in the command syntax statements.

Some CP/M 3 commands accept wildcards in the filename and filetype parts of the command tail. For example,

```
B:MY*.A??
```

is a file specification with drive specifier B:, filename MY*, and filetype A??. This ambiguous file specification might match several files in the directory.

Put together, the parts of a file specification are represented like this:

d:filename.typ;password

In the preceding form, d: represents the optional drive specifier, filename represents the one- to eight-character filename, and typ represents the optional one- to three-character filetype. The syntax descriptions in this section use the term filespec to indicate any valid combination of the elements included in the file specification. The following list shows valid combinations of the elements of a CP/M 3 file specification.

- filename
- filename.typ
- filename;password
- filename.typ;password
- d:filename
- d:filename.typ
- d:filename;password
- d:filename.typ;password

The characters in Table 5-1 have special meaning in CP/M 3, so do not use these characters in file specifications except as indicated.

**Table 5-1. Reserved Characters**

| *Character* | *Meaning* |
|---|---|
| < = , ! \| > [ ]<br>tab space<br>carriage return | file specification delimiters |
| : | drive delimiter in file specification |
| . | filetype delimiter in file specification |
| ; | password delimiter in file specification |
| * ? | wildcard characters in an ambiguous file specification |
| < > & ! \| \ + – | option list delimiters |
| [ ] | option list delimiters for global and local options |
| ( ) | delimiters for multiple modifiers inside square brackets for options that have modifiers |
| / $ | option delimiters in a command line |
| ; | comment delimiter at the beginning of a command line |

CP/M 3 has already established several file groups. Table 5-2 lists some of their filetypes with a short description of each family. Appendix C provides the complete list.

**Table 5-2. CP/M 3 Filetypes**

| *Filetype* | *Meaning* |
|---|---|
| ASM | Assembler source file |
| BAS | CBASIC® source program |
| COM | 8080, 8085, or equivalent machine language program |
| HLP | HELP message file |
| SUB | List of commands to be executed by SUBMIT |
| $$$ | Temporary file |

In some commands, descriptive qualifiers are used with filespecs to further qualify the type of filespec accepted by the commands. For example, wildcard-filespec denotes wildcard specifications, dest-filespec denotes a destination filespec, and src-filespec denotes a source filespec.

You now understand command keywords, command tails, control characters, default drive, and wildcards. You also see how to use the formal names filespec, drive specifier, filename, and filetype. These concepts give you the background necessary to compose complete command lines.

## 5.2 How Commands Are Described

CP/M 3 commands appear in alphabetical order. Each command description is given in a specific form. This section also describes the notation that indicates the optional parts of a command line and other syntax notation.

- The description begins with the command keyword in upper-case.
- The syntax section gives you one or more general forms to follow when you compose the command line.

- The explanation section defines the general use of the command keyword, and points out exceptions and special cases. The explanation sometimes includes tables or lists of options that you can use in the command line.
- The examples section lists a number of valid command lines that use the command keyword. To clarify examples of interactions between you and the operating system, the characters that you enter are slanted. The responses that CP/M 3 shows on your screen are in vertical type.

The notation in the syntax lines describes the general command form using these rules:

- Words in capital letters must be spelled as shown, but you can use any combination of upper- or lower-case letters.
- The symbolic notation d:, filename, .typ, ;password, and filespec have the general meanings described in Section 5.1.
- You must include one or more space characters where a space is shown, unless otherwise specified. For example, the PIP options do not need to be separated by spaces.

The following table defines the special symbols and abbreviations used in syntax lines.

**Table 5-3. Syntax Notation**

| *Symbol* | *Meaning* |
|---|---|
| DIR | Directory attribute. |
| n | You can substitute a number for n. |
| o | Indicates an option or an option list. |
| RO | Read-Only. |
| RW | Read-Write. |
| s | You can substitute a string, which consists of a group of characters, for s. |

Table 5-3. (continued)

| *Symbol* | *Meaning* |
|---|---|
| SYS | System attribute. |
| {} | Items within braces are optional. You can enter a command without the optional items. The optional items add effects to your command line. |
| [] | Items in square brackets are options or an option list. If you use an option specified within the brackets, you must type the brackets to enclose the option. If the right bracket is the last character on the command line, it can be omitted. |
| () | Items in parentheses indicate a range of options. If you use a range from an option list, you must enclose the range in parentheses. |
| ... | Ellipses tell you that the previous item can be repeated any number of times. |
| \| | The or bar separates alternative items in a command line. You can select any or all of the alternatives specified. Mutually exclusive options are indicated in additional syntax lines or are specifically noted in the text. |
| ↑ or CTRL | Represent the CTRL key on your keyboard. (CTRL characters show as ^ on your screen.) |
| <cr> | Indicates a carriage return keystroke. |
| * | Wildcard character—replaces all or part of a filename and/or filetype. |
| ? | Wildcard character—replaces any single character in the same position of a filename or filetype. |

Let's look at some examples of syntax notation. The CP/M 3 DIR (DIRectory) command displays the names of files cataloged in the disk directory and, optionally, displays other information about the files.

The syntax of the DIR command with options shows how to use the command line syntax notation:

```
Syntax:     DIR {d:} | {filespec} {[options]}
                  |         |          |
              optional  optional   optional
```

This tells you that the command tail following the command keyword DIR is optional. DIR alone is a valid command, but you can include a file specification, or a drive specification, or just the options in the command line. Therefore,

```
DIR
DIR filespec
DIR d:
DIR [RO]
```

are valid commands. Furthermore, the drive or file specification can be followed by another optional value selected from one of the following list of DIR options:

```
RO
RW
DIR
SYS
```

Therefore,

```
DIR d:filespec [RO]
```

is a valid command.

Recall that in Section 2 you learned about wildcards in filenames and filetypes. The DIR command accepts wildcards in the file specification.

Using this syntax, you can construct several valid command lines:

DIR
DIR X.PAS
DIR X.PAS [RO]
DIR X.PAS [SYS]
DIR *.PAS
DIR *.* [RW]
DIR X.* [DIR]

The CP/M 3 command PIP (Peripheral Interchange Program) is the file copy program. PIP can copy information from the disk to the screen or printer. PIP can combine two or more files into one longer file. PIP can also rename files after copying them. Look at one of the formats of the PIP command line for another example of how to use command line notation. PIP also copies files from one disk to another disk.

Syntax: PIP dest-filespec = src-filespec{,filespec...}

In the preceding example, dest-filespec is further defined as a destination file specification or peripheral device (printer, for example) that receives data. Similarly, src-filespec is a source file specification or peripheral device (keyboard, for example) that transmits data. PIP accepts wildcards in the filename and filetype. (See the PIP command for details regarding other capabilities of PIP.) There are, of course, many valid command lines that come from this syntax. Some examples follow.

```
PIP NEWFILE.DAT=OLDFILE.DAT
PIP B:=A:THISFILE.DAT
PIP B:X.BAS=Y.BAS, Z.BAS
PIP X.BAS=A.BAS, B.BAS, C.BAS
PIP B:=A:*.BAK
PIP B:=A:*.*
```

The remainder of this section contains a complete description of each CP/M 3 utility. The descriptions are arranged alphabetically for easy reference.

# The COPYSYS Command

**Syntax:** COPYSYS

**Explanation:** The COPYSYS command copies the CP/M 3 system from a CP/M 3 system disk to another disk. The disk must have the same format as the original system disk. For example, if the system disk is a single-density disk, the disk you use to copy onto must also be in single-density format.

The COPYSYS utility copies only the system tracks onto the new disk. To use the new disk as a CP/M 3 system disk, you must also copy the system file CPM3.SYS to the new disk. COPYSYS gives you the option to copy CPM3.SYS to the new disk. To copy other files onto the new disk, use the PIP command.

**Example:**

```
A>COPYSYS
Copysys Ver 3.0
Source drive name (or return for default)C
Source on C then type return
```

Place the disk to be copied in drive C, then enter <cr>.

```
Function Complete
Destination drive name (or return to reboot)C
Destination on C then type return
```

Replace the system disk in C with the new disk, then enter <cr>.

```
Function complete
Do you wish to copy CPM3.SYS?Y
Source drive name (or return for default)<cr>
Source on default then type return
Function complete
Destination drive name (or return to reboot)C
Destination on C then type return
```

```
Place the disk to be copied in drive C then enter <cr>.

Function complete
```

The preceding example copies the CP/M 3 system using only one disk drive C. In the preceding messages, the word source refers to the disk that contains the CP/M 3 system, and the word destination refers to the disk to which the CP/M 3 system is to be copied.

The system file CPM3.SYS is copied from the default drive A to the new disk in drive C. CP/M 3 requires the file CPM3.SYS to be on the system disk.

# The DATE Command

**Syntax:** DATE {CONTINUOUS}
DATE {time-specification}
DATE SET

**Explanation:** The DATE command is a transient utility that lets you display and set the date and time of day. When you start CP/M 3, the date and time are set to the creation date of your CP/M 3 system. Use DATE to change this initial value to the current date and time.

## Display Current Date and Time

**Syntax:** DATE {CONTINUOUS}

**Explanation:** The preceding form of the DATE command displays the current date and time. The CONTINUOUS option allows continuous display of the date and time. The CONTINUOUS option can be abbreviated to C. You can stop the continuous display by pressing any key.

**Examples:**

```
A>DATE
A>DATE C
```

The first example displays the current date and time. A sample display might be:

```
Fri 08/13/82 09:15:37
```

The second example displays the date and time continuously until you press any key to stop the display.

## Set the Date and Time

**Syntax:** DATE {time-specification}
DATE SET

**Explanation:** The first form allows the user to enter both date and time in the command. The time-specification format is

MM/DD/YY HH:MM:SS

where:

MM is a month value in the range 1 to 12.
DD is a day value in the range 1 to 31.
YY is the two-digit year value relative to 1900.
HH is the hour value in the range of 0 to 23.
MM is the minute value in the range of 0 to 59.
SS is the second value in the range of 0 to 59.

The system checks the validity of the date and time entry and determines the day for the date entered.

The second form prompts you to enter the date and the time. To keep the current system date or time, press the carriage return.

**Examples:**

```
A>DATE 08/14/82 10:30:00
```

The system responds with

```
Press any key to set time
```

When the time occurs, press any key. DATE initializes the time at that instant, and displays the date and time:

```
Sat 08/14/82 10:30:00
```

```
A>DATE SET
```

The system prompts with

```
Enter today's date (MM/DD/YY):
```

Press the carriage return to skip or enter the date. Then the system prompts with

```
Enter the time (HH:MM:SS):
```

Press the carriage return to skip or enter the time and the system prompts with

```
Press any Key to set time
```

to allow you to set the time exactly.

# The DEVICE Command

**Syntax:**

DEVICE {NAMES | VALUES | physical-dev | logical-dev}
DEVICE logical-dev = physical-dev {option}
{,physical-dev {option},...}
DEVICE logical-dev = NULL
DEVICE physical-dev {option}
DEVICE CONSOLE [PAGE | COLUMNS = columns | LINES = lines}

**Explanation:** The DEVICE command is a transient utility that displays current assignments of CP/M 3 logical devices and the names of physical devices. DEVICE allows you to assign logical CP/M 3 devices to peripheral devices attached to the computer. The DEVICE command also sets the communications protocol and speed of a peripheral device, and displays or sets the current console screen size.

CP/M 3 supports the following five logical devices:

CONIN:
CONOUT:
AUXIN:
AUXOUT:
LST:

These logical devices are also known by the following names:

CON: (for CONIN: and CONOUT:)
CONSOLE: (for CONIN: and CONOUT:)
KEYBOARD (for CONIN:)
AUX: (for AUXIN: and AUXOUT:)
AUXILIARY: (for AUXIN: and AUXOUT:)
PRINTER (for LST:)

The physical device names on a computer vary from system to system. You can use the DEVICE command to display the names and attributes of the physical devices that your system accepts.

## Display Device Characteristics and Assignments

**Syntax:** DEVICE { NAMES | VALUES | physical-dev | logical-dev}

**Explanation:** The preceding form of the DEVICE command displays the names and attributes of the physical devices and the current assignments of the logical devices in the system.

**Examples:** *A>DEVICE*

The preceding command displays the physical devices and current assignments of the logical devices in the system. The following is a sample response:

```
Physical Devices:
I=Input,O=Output,S=Serial,X=Xon-Xoff
CRT     9600  IOS     LPT     9600  IOSX    CRT1    9600    IOS
CRT2    9600  IOS     CRT3    4800  IOS     LPT1    134     IOSX
CEN     NONE   O      MODEM1  19200 IOS     MODEM2  300       S
CTRLR1  150    O      GRACRT  19200 IOS     DIABLO  110      O
CTRLR2  300    O      SCRTY   7200

Current Assignments:
CONIN:   = CRT
CONOUT:  = CRT
AUXIN:   = Null Device
AUXOUT:  = Null Device
LST:     = LPT

Enter new assignment or hit RETURN:
```

The system prompts for a new device assignment. You can enter any valid device assignment (as described in the next section). If you do not want to change any device assignments, press the RETURN key.

*A>DEVICE NAMES*

The preceding command lists the physical devices with a summary of the device characteristics.

```
A>DEVICE VALUES
```

The preceding command displays the current logical device assignments.

```
A>DEVICE CRT
```

The preceding command displays the attributes of the physical device CRT.

```
A>DEVICE CON
```

The preceding command displays the assignment of the logical device CON:

## Assign a Logical Device

**Syntax:** DEVICE logical-dev = physical-dev {option}
{,physical-dev {option},...}
DEVICE logical-dev = NULL

**Explanation:** The first form assigns a logical device to one or more physical devices. The second form disconnects the logical device from any physical device.

**Table 5-4. DEVICE Options**

| *Option* | *Meaning* |
| --- | --- |
| XON | refers to the XON/XOFF communications protocol. This protocol uses two special characters in the ASCII character set called XON and XOFF. XON signals transmission on, and XOFF signals transmission off. Before each character is output from the computer to the peripheral device, the computer checks to see if there is any incoming data from the peripheral. If the incoming character is XOFF, the computer suspends all further output until it receives an XON from the device, indicating that the device is again ready to receive more data. |
| NOXON | indicates no protocol and the computer sends data to the device whether or not the device is ready to receive it. |
| baud-rate | is the speed of the device. The system accepts the following baud rates:<br><br>50 75 110 134<br>150 300 600 1200<br>1800 2400 3600 4800<br>7200 9600 19200 |

**Examples:**

```
A>DEVICE CONOUT:=LPT,CRT
A>DEVICE AUXIN:=CRT2 [XON,9600]
A>DEVICE LST:=NULL
```

The first example assigns the system console output, CONOUT:, to the printer, LPT, and the screen, CRT. The second example assigns the auxiliary logical input device, AUXIN:, to the physical device CRT using protocol XON/XOFF and sets the transmission rate for the device at 9600. The third example disconnects the list output logical device, LST:.

## Set Attributes of a Physical Device

**Syntax:** DEVICE physical-dev {option}

**Explanation:** The preceding form of the DEVICE command sets the attributes of the physical device specified in the command.

**Example:**

```
A>DEVICE LPT [XON,9600]
```

The preceding command sets the XON/XOFF protocol for the physical device LPT and sets the transmission speed at 9600.

## Display or Set the Current Console Screen Size

**Syntax:** DEVICE CONSOLE [PAGE | COLUMNS = columns | LINES = lines]

**Explanation:** The preceding form of the DEVICE command displays or sets the current console size.

**Examples:**

```
A>DEVICE CONSOLE [PAGE]
A>DEVICE CONSOLE [COLUMNS=40, LINES=16]
```

The first example displays the current console page width in columns and length in lines. The second example sets the screen size to 40 columns and 16 lines.

# The DIR Command

**Syntax:** DIR {d:}
DIR {filespec}

DIRSYS {d:}
DIRSYS {filespec}

DIR {d:} [options]
DIR {filespec} {filespec}...[options]

**Explanation:** The DIR command displays the names of files and the attributes associated with the files. DIR and DIRSYS are built-in utilities; DIR with options is a transient utility.

## Display Directory

**Syntax:** DIR {d:}
DIR {filespec}

DIRSYS {d:}
DIRSYS {filespec}

**Explanation:** The DIR and DIRSYS commands display the names of files cataloged in the directory of an on-line disk. The DIR command lists the names of files in the current user number that have the Directory (DIR) attribute. DIR accepts wildcards in the file specification. You can abbreviate the DIRSYS command to DIRS.

The DIRSYS command displays the names of files in the current user number that have the System (SYS) attribute. Although you can read System (SYS) files that are stored in user 0 from any other user number on the same drive, DIRSYS only displays user 0 files if the current user number is 0. DIRSYS accepts wildcards in the file specification.

If you omit the drive and file specifications, the DIR command displays the names of all files with the DIR attribute on the default drive for the current user number. Similarly, DIRSYS displays all the SYS files.

If the drive specifier is included, but the filename and filetype are omitted, the DIR command displays the names of all DIR files in the current user on the disk in the specified drive. DIRSYS displays the SYS files.

If the file specification contains wildcard characters, all filenames that satisfy the match are displayed on the screen.

If no filenames match the file specification, or if no files are cataloged in the directory of the disk in the named drive, the DIR or DIRSYS command displays the message:

```
No File
```

If system (SYS) files match the file specification, DIR displays the message:

```
SYSTEM FILE(S) EXIST
```

If nonsystem (DIR) files match the file specification, DIRSYS displays the message:

```
NON-SYSTEM FILES(S) EXIST
```

The DIR command pauses after filling the screen. Press any key to continue the display.

**Note:** You can use the DEVICE command to change the number of columns displayed by DIR or DIRSYS.

**Examples:** `A>`*`DIR`*

Displays all DIR files cataloged in user 0 on the default drive A.

`A>`*`DIR B:`*

Displays all DIR files for user 0 on drive B.

`A>`*`DIR B:X.BAS`*

Displays the name X.BAS if the file X.BAS is present on drive B.

```
4A>DIR *.BAS
```

Displays all DIR files with filetype BAS for user 4 on drive A.

```
B>DIR A:X*.C?D
```

Displays all DIR files for user 0 on drive A whose filename begins with the letter X, and whose three character filetype contains the first character C and last character D.

```
A>DIRSYS
```

Displays all files for user 0 on drive A that have the system (SYS) attribute.

```
3A>DIRS *.COM
```

This abbreviated form of the DIRSYS command displays all SYS files with filetype COM on the default drive A for user 3.

## Display Directory with Options

**Syntax:** DIR {d:} [options]
DIR {filespec} {filespec}...[options]

**Explanation:** The DIR command with options is an enhanced version of the DIR command. The DIR command displays CP/M 3 files in a variety of ways. DIR can search for files on any or all drives, for any or all user numbers.

DIR allows the option list to occur anywhere in the command tail. These options modify the entire command line. Only one option list is allowed.

Options must be enclosed in square brackets. The options can be used individually, or strung together separated by commas or spaces. Options can be abbreviated to only one or two letters if the abbreviation unambiguously identifies the option.

If a directory listing exceeds the size of your screen, DIR automatically halts the display when it fills the screen. Press any key to continue the display.

**Table 5-5. DIR Display Options**

| *Option* | *Function* |
|---|---|
| ATT | displays the user-definable file attributes F1, F2, F3, and F4. |
| DATE | displays files with date and time stamps. If date and time stamping is not active, DIR displays the message: `Date and Time Stamping Inactive.` |
| DIR | displays only files that have the DIR attribute. |
| DRIVE=ALL | displays files on all accessed drives. DISK is also acceptable in place of DRIVE in all the DRIVE options. |
| DRIVE=(A,B,C,...,P) | displays files on the drives specified. |
| DRIVE=d | displays files on the drive specified by d. |

Table 5-5. (continued)

| *Option* | *Function* |
|---|---|
| EXCLUDE | displays the files on the default drive and user area that do not match the files specified in the command line. |
| FF | sends an initial form-feed to the printer device if the printer has been activated by CTRL-P. If the LENGTH=n option is also specified, DIR issues a form-feed every n lines. Otherwise, the FF option deactivates the default paged output display. |
| FULL | shows the name of the file and the size of the file. The size is shown as the amount of space in kilobytes and the number of 128-byte records allocated to the file. FULL also shows the attributes the file. (See the SET command for description of file attributes). If there is a directory label on the drive, DIR shows the password protection mode and the time stamps. The display is alphabetically sorted. FULL is the default output format for display when using DIR with options. |
| LENGTH=n | displays n lines of output before inserting a table heading. n must be in the range between 5 and 65536. The default length is one full screen of information. |

Table 5-5. (continued)

| *Option* | *Function* |
|---|---|
| MESSAGE | displays the names of the specified drives and user numbers it is currently searching. If there are no files in the specified locations, DIR displays the file not found message. |
| NOPAGE | continuously scrolls information by on the screen. Does not wait for you to press a key to restart the scrolling movement. |
| NOSORT | displays files in the order it finds them on the disk. If this option is not included, DIR displays the files alphabetically. |
| RO | displays only the files that have the Read-Only attribute. |
| RW | displays only the files that are set to Read-Write. |
| SIZE | displays the filename and file size in kilobytes. |
| SYS | displays only the files that have the SYS attribute. |

Table 5-5. (continued)

| *Option* | *Function* |
|---|---|
| USER = ALL | displays all files under all the user numbers for the default drive. |
| USER = n | displays the files under the user number specified by n. |
| USER = (0,1,...,15) | displays files under the user numbers specified. |

**Examples:** *A>DIR C: [FULL]*
*A>DIR C: [SIZE]*

The following is sample output of the [FULL] option display format shown in the first example of the DIR command:

```
Directory for Drive C:  User  0

    Name     Bytes   Recs  Attributes  Prot      Update            Access
------------ ----- ------- ---------- ------ --------------- ---------------
DITS     BAK    1k       1 Dir RW     Read   09/01/82 13:04  09/01/82 13:07
DITS     TES    1k       1 Dir RO     None   09/01/82 13:07  09/01/82 13:09
DITS     Y      1k       1 Dir RW     None   08/25/82 03:33  08/25/82 03:33
DITS     ZZ     1k       1 Dir RW     None   08/25/82 03:36  08/25/82 03:36
SETDEF   COM    4k      29 Dir RO     None                   08/25/82 03:36
SUBMIT   TX2    1k       1 Dir RO     None
SUBMIT   TX1    5k      43 Dir RO     None

Total Bytes     =    14k  Total Records =     77  Files Found =     7
Total 1k Blocks =    14   Used/Max Dir Entries for Drive C:   11/  64
```

The following is sample output of the [SIZE] option display format shown in the second example of the DIR command:

```
Directory for Drive C:  User  0

C: DITS     BAK     1k : DITS     TES     1k : DITS     Y       1k
C: DITS     ZZ      1k : SETDEF   COM     4k : SUBMIT   TX2     1k
C: SUBMIT   TX1     5k :

Total Bytes     =       14k  Total Records =      77  Files Found =     7
Total 1k Blocks =       14   Used/Max Dir Entries for Drive C:    11/  64
```

Both the full format and the size format follow their display with two lines of totals. The first line displays the total number of kilobytes, the total number of records, and the total number of files for that drive and user area. The second line displays the total number of 1K blocks needed to store the listed files. The number of 1K blocks shows the amount of storage needed to store the files on a single density disk, or on any drive that has a block size of one kilobyte. The second line also shows the number of directory entries used per number of directory entries available on the drive.

*A>DIR [DRIVE=C,FF]*

DIR sends a form-feed to the printer before displaying the files on drive C.

*A>DIR D: [RW,SYS]*

The preceding example displays all the files on drive D with Read-Write and SYS attributes.

*A>DIR C: [USER=ALL]*

Displays all the files under each user number (0-15) on drive C.

*A>DIR [USER=2]*

Displays all the files under user 2 on the default drive.

```
A>DIR C: [USER=(3,4,10)]
```

This example displays all the files under user numbers 3, 4, and 10 on drive C.

```
A>DIR [DRIVE=ALL]
```

Displays all the files under user 0 on all the drives in the drive search chain. (See the SETDEF command.)

```
4A>DIR [DRIVE=C]
```

Displays all the files under user 4 on drive C.

```
A>DIR [DRIVE=(B,D)]
```

Displays all the files under user 0 on drives B and D.

```
A>DIR [exclude] *.COM
```

The preceding example above lists all the files on the default drive and user 0 that do not have a filetype of COM.

```
A>DIR [user=all,drive=all,sys] *.PLI *.COM *.ASM
```

The preceding command line instructs DIR to list all the system files of type PLI, COM, and ASM on the system in the currently active drives for all the user numbers on the drives.

```
A>DIR X.SUB [MESSAGE,USER=ALL,DRIVE=ALL]
```

The preceding command searches all drives under each user number for X.SUB. During the search, DIR displays the drives and user numbers.

```
A>DIR [drive=all user=all] TESTFILE.BOB
```

The preceding example instructs DIR to display the filename TESTFILE.BOB if it is found on any logged-in drive for any user number.

```
A>DIR [size,rw]D:
```

The preceding example instructs DIR to list each Read-Write file that resides on drive D with its size in kilobytes. Note that D: is equivalent to D:*.*.

# The DUMP Command

**Syntax:** DUMP filespec

**Explanation:** Dump displays the contents of a file in hexadecimal and ASCII format.

**Example:** *A>DUMP ABC.TEX*

Console output can look like the following:

```
DUMP - Version 3.0
0000: 41 42 43 0D 0A 44 45 46 0D 0A 47 48 49 0D 0A 1A  ABC..DEF..GHI....
0010: 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A  ................
   .
   .
   .
```

# The ED Command

**Syntax:** ED {input-filespec {d: | output-filespec}}

**Explanation:** The ED transient utility lets you create and edit a disk file.

The ED utility is a line-oriented context editor. This means that you create and change character files line-by-line, or by referencing individual characters within a line.

The ED utility lets you create or alter the file named in the file specification. Refer to Section 6 for a description of the ED utility.

The ED utility uses a portion of your user memory as the active text buffer where you add, delete, or alter the characters in the file. You use the A command to read all or a portion of the file into the buffer. You use the W or E command to write all or a portion of the characters from the buffer back to the file.

An imaginary character pointer, called CP, is at the beginning of the buffer, between two characters in the buffer, or at the end of the buffer.

You interact with the ED utility in either command or insert mode. ED displays the * prompt on the screen when ED is in command mode. When the * appears, you can enter the single letter command that reads text from the buffer, moves the CP, or changes the ED mode of operation. When in command mode, you can use the line-editing characters CTRL-C, CTRL-E, CTRL-H, CTRL-U, CTRL-X, and RUBOUT to edit your input. In insert mode, however, you use only CTRL-H, CTRL-U, CTRL-X, and RUBOUT.

Table 5-6. ED Command Summary

| *Command* | *Action* |
|---|---|
| nA | Append n lines from original file to memory buffer. |
| 0A | Append file until buffer is one half full. |
| #A | Append file until buffer is full (or end of file). |
| B, -B | Move CP to beginning (B) or bottom (-B) of buffer. |
| nC, -nC | Move CP n characters forward (C) or back (-C) through buffer. |
| nD, -nD | Delete n characters before (-D) or from (D) the CP. |
| E | Save new file and return to CP/M 3. |
| Fstring{↑Z} | Find character string. |

Table 5-6. (continued)

| *Command* | *Action* |
|---|---|
| `H` | Save the new file, then reedit, using the new file as the original file. |
| `I` | Enter insert mode; use ↑Z or ESCape to exit insert mode. |
| `Istring{↑Z}` | Insert string at CP. |
| **Note:** upper-case I forces all input to upper-case; while lower-case i allows upper- and lower-case. | |
| `Jsearch_str^Zins_str^Zdel_to_str{↑Z}` | Juxtapose strings. |
| `nK, -nK` | Delete (kill) n lines from the CP. |
| `nL, -nL, 0L` | Move CP n lines. |
| `nMcommands` | Execute commands n times. |
| `n, -n` | Move CP n lines and display that line. |

Table 5-6. (continued)

| *Command* | *Action* |
|---|---|
| n: | Move to line n. |
| :ncommand | Execute command through line n. |
| Nstring{↑Z} | Extended find string. |
| O | Return to original file. |
| nP, -nP | Move CP n lines forward and display n lines at console. |
| Q | Abandon new file, return to CP/M 3. |
| R{↑Z} | Read X$$$$$$$.LIB file into buffer. |
| Rfilespec{↑Z} | Read filespec into buffer. |
| Sdelete string^Zinsert string{↑Z} | Substitute string. |

Table 5-6. (continued)

| *Command* | *Action* |
|---|---|
| nT, -nT, 0T | Type n lines. |
| U, -U | Upper-case translation. |
| V, -V, 0V | Line numbering on/off, display free buffer space. |
| nW | Write n lines to updated file. |
| nX{↑Z} | Write or append n lines to X$$$$$$$.LIB. |
| nXfilespec{↑Z} | Write n lines to filespec or append if previous x command applied to the same file. |
| 0X{↑Z} | Delete file X$$$$$$$.LIB. |
| 0Xfilespec{↑Z} | Delete filespec. |
| nZ | Wait n seconds. |

Section 6 gives a detailed description of the overall operation of the ED utility and the use of each command.

If you do not include a command tail in the ED command, it prompts you for the input filespec and the output filespec as follows:

```
Enter Input File:
```

After you enter the input filespec, ED prompts again:

```
Enter Output File:
```

Enter a filename or drive if you want the output file or its location to be different from that of the input file. Press RETURN if you want the output file to replace the input file. In this case, the input file is renamed to type BAK.

If the second file specification contains only the drive specifier, the second filename and filetype become the same as the first filename and filetype.

If the file given by the first file specification is not present, ED creates the file and writes the message:

```
NEW FILE
```

If the file given by the first filespec is already present, you must issue the A command to read portions of the file to the buffer. If the size of the file does not exceed the size of the buffer, the command

```
*a
```

reads the entire file to the buffer.

The i (Insert) command places ED in insert mode. In this mode, any characters you type are stored in sequence in the buffer starting at the current CP.

Any single letter commands typed in insert mode are not interpreted as commands, but are simply stored in the buffer. To return from insert mode to command mode, press CTRL-Z or the ESC key. Note that you can always substitute the ESC key for CTRL-Z in ED.

The single letter commands are usually typed in lower-case. The commands that must be followed by a character sequence end with CTRL-Z if they are to be followed by another command letter.

Any single letter command typed in upper-case tells ED to internally translate to upper-case all characters up to the CTRL-Z that ends the command.

When enabled, line numbers that appear on the left of the screen take the form:

nnnnn:

where nnnnn is a number in the range 1 through 65535. Line numbers are displayed for your reference and are not contained in either the buffer or the character file. The screen line starts with

```
:*
```

when the CP is at the beginning or end of the buffer.

**Examples:**

```
A>ED MYPROG.PAS
```

If not already present, this command line creates the file MYPROG.PAS on drive A. The command prompt

```
:*
```

appears on the screen. This tells you that the CP is at the beginning of the buffer. If the file is already present, issue the command

```
:*#a
```

to fill the buffer. Then type the command

```
:*0p
```

to fill the screen with the first n lines of the buffer, where n is the current default page size (See the DEVICE command to set the page size).

Type the command

```
:*e
```

to stop the ED utility when you are finished changing the character file. The ED utility leaves the original file unchanged as MYPROG.BAK and the altered file as MYPROG.PAS.

```
A>ED MYPROG.PAS B:NEWPROG.PAS
```

The original file is MYPROG.PAS on the default drive A. The original file remains unchanged when the ED utility finishes, with the altered file stored as NEWPROG.PAS on drive B.

```
A>B:ED MYPROG.PAS B:
```

The ED.COM file must be on drive B. The original file is MYPROG.PAS located on drive A. It remains unchanged, with the altered program stored on drive B as MYPROG.PAS.

# The ERASE Command

**Syntax:** ERASE {filespec} {[CONFIRM]}

**Explanation:** The ERASE command removes one or more files from a disk's directory in the current user number. Wildcard characters are accepted in the filespec. Directory and data space are automatically reclaimed for later use by another file. The ERASE command can be abbreviated to ERA.

Use the ERASE command with care because all files in the current user number that satisfy the file specification are removed from the disk directory.

Command lines that take the form

ERASE {d:}wildcard-filespec

require your confirmation because they erase an entire group of files, not just one file. The system prompts with the following message:

```
ERASE {d:}wildcard-filespec (Y/N)?
```

Respond with y if you want to remove all matching files, and n if you want to avoid erasing any files.

If no files match the file specification, you see the following message:

```
No File
```

The CONFIRM option informs the system to prompt for verification before erasing each file that matches the filespec. You can abbreviate CONFIRM to C.

If you use the CONFIRM option with wildcard-filespec, then ERASE prompts for confirmation for each file. You can selectively erase the files you want by responding Y to the confirm message, or keep the files by responding N to the confirm message.

**Examples:**

```
A>ERASE X.PAS
```

This command removes the file X.PAS from the disk in drive A.

```
A>ERA *.PRN
```

The system asks to confirm:

```
ERASE *.PRN (Y/N)?Y
```

All files with the filetype PRN are removed from the disk in drive A.

```
B>ERA A:MY*.* [CONFIRM]
```

Each file on drive A with a filename that begins with MY is displayed with a question mark for confirmation. Type Y to erase the file displayed, N to keep the file.

```
A>ERA B:*.*
ERASE B:*.* (Y/N)?Y
```

All files on drive B are removed from the disk.

# The GENCOM Command

**Syntax:** GENCOM {COM-filespec} {RSX-filespec}...
{[LOADER|NULL|SCB = (offset,value)]}

**Explanation:** The GENCOM command is a transient utility that creates a special COM file with attached RSX files. RSX files are used as Resident System Extensions and are discussed in detail in the *CP/M Plus (CP/M Version 3) Operating System Programmer's Guide*. GENCOM places a special header at the beginning of the output program file to indicate to the system that RSX loading is required. It can also set a flag to keep the program loader active.

The GENCOM command can also restore a file already processed by GENCOM to the original COM file without the header and RSXs. GENCOM has three options that help you attach RSX files:

- The LOADER option sets a flag to keep the program loader active. (For complete details on the LOADER option read about CP/M function 59 in the *CP/M Plus (CP/M Version 3) Operating System Programmer's Guide*.) This option is used only if no RSX files are attached to the COM file.
- The NULL option indicates that only RSX files are specified. GENCOM creates a dummy COM file for the RSX files. The output COM filename is taken from the filename of the first RSX-filespec.
- The SCB = (offset,value) option sets the System Control Block from the program by using the hex values specified by (offset,value). For complete details on the SCB option read about CP/M function 49 in the *CP/M Plus (CP/M Version 3) Operating System Programmer's Guide*.

## Attach RSX Files to a COM File

**Syntax:** GENCOM COM-filespec RSX-filespec...
{[LOADER|SCB = (offset,value)]}

**Explanation:** The preceding form of the GENCOM command creates a COM file with a header and attached RSXs. A maximum of 15 RSXs can be attached. GENCOM expects the first filespec to be a COM file and the following filespecs to be RSX files. Note that the original COM file is replaced by the newly-created COM file.

**Example:** `A>GENCOM MYPROG PROG1 PROG2`

The preceding command generates a new COM file MYPROG.COM with attached RSXs PROG1 and PROG2.

## Generate a COM File Using only RSX Files

**Syntax:** GENCOM RSX-filespec {RSX-filespec}...
[NULL {SCB = (offset,value)}]

**Explanation:** The preceding form of the GENCOM command attaches the RSX files to a dummy COM file. GENCOM creates a COM file with the filename of the first RSX-filespec in the command tail. This format allows the system to load RSXs directly.

**Example:** `A>GENCOM PROG1 PROG2 [NULL]`

The preceding command creates a COM file PROG1.COM with Resident System Extensions PROG1.RSX and PROG2.RSX.

## Restore a File with Attached RSXs to Original COM File

**Syntax:** GENCOM filename

**Explanation:** The preceding form of the GENCOM file takes a file that has already been processed by GENCOM and restores it to its original COM file format. This form of the command assumes a filetype of COM.

**Example:** `A>GENCOM MYPROG`

In the preceding command, GENCOM takes MYPROG.COM, strips off the header and deletes all attached RSXs to restore it to its original COM format.

## Update (Add or Replace) RSX Files

**Syntax:** GENCOM COM-filespec RSX-filespec...
{[LOADER | SCB=(offset,value)]}

**Explanation:** The preceding form of the GENCOM command adds and/or replaces RSX files to a file already processed by GENCOM.

GENCOM inspects the list of RSX files. If they are new, they are added to the file already processed by GENCOM. If they already exist, then GENCOM replaces the existing RSXs with the new RSX files.

**Example:** *A>GENCOM MYPROG PROG1 PROG2*

In the preceding example, GENCOM looks at MYPROG.COM, which is already processed by GENCOM, to see if PROG1.RSX and PROG2.RSX are already attached RSX files in the module. If either one is already attached, GENCOM replaces it with the new RSX module. Otherwise, GENCOM appends the specified RSX files to the COM file.

## Attach a Header Record

**Syntax:** GENCOM filename [SCB=(offset,value),... | LOADER]

**Explanation:** The preceding syntax line attaches a GENCOM header record, with the SCB or loader flag set, to a file of type COM that contains no RSXs. This form of the command does not attach RSXs to a file.

**Examples:** `A>GENCOM FILETWO [loader]`

The preceding command attaches a 256-byte header record to the file FILETWO.COM and sets the loader flag in the header record.

`A>GENCOM FILEFOUR [scb=(1,1)]`

The preceding command causes the program loader to set byte 1 of the System Control Block to 1 when it loads FILEFOUR.COM.

For more information, see functions 49, Set/Get System Control Block, and 59, Load Overlay or Resident System Extensions, in the *CP/M Plus (CP/M Version 3) Operating System Programmer's Guide.*

# The GET Command

**Syntax:** GET {CONSOLE INPUT FROM} FILE filespec
{[{ECHO|NO ECHO}|SYSTEM]}
GET {CONSOLE INPUT FROM} CONSOLE

**Explanation:** The GET command is a transient utility that directs CP/M 3 to take console input from a file. The file can contain CP/M 3 system commands and/or input for a user program. If you use the SYSTEM option, GET immediately takes the next system command from the file.

Console input is taken from a file until the program terminates. If the file is exhausted before program input is terminated, the program looks for subsequent input from the console. If the program terminates before exhausting all its input, the system reverts back to the console for console input.

When the SYSTEM option is used, the system immediately goes to the file specified for console input. If you omit the SYSTEM option, you can enter one system command to initiate a user program whose console input is taken from the file specified in the GET command. The system reverts to the console for input when it reaches the end of the GET file input. The system also reverts to the console for console input if a GET CONSOLE INPUT FROM CONSOLE command is included in the input file.

## Get Console Input from a File

**Syntax:** GET {CONSOLE INPUT FROM} FILE filespec {[options]}

**Explanation:** The preceding form of the GET command tells the system to get subsequent console input from a file. Table 5-7 lists the GET options that you use in the following format:

[{ECHO | NO ECHO} | SYSTEM]

**Table 5-7. GET Options**

| *Option* | *Meaning* |
|---|---|
| ECHO | specifies that the input is echoed to the console. This is the default option. |
| NO ECHO | specifies that the file input is not to be echoed to the console. The program output and the system prompts are not affected by this option and are still echoed to the console. |
| SYSTEM | specifies that all system input is to be taken from the disk file specified in the command line. GET takes system and program input from the file until the file is exhausted or until GET reads a GET console command from the file. |

**Examples:**

```
A>GET FILE XINPUT
A>MYPROG
```

The preceding sequence of commands tells the system to activate the GET utility. However, because SYSTEM is not specified, the system reads the next input line from the console and executes MYPROG. If MYPROG program requires console input, it is taken from the file XINPUT. When MYPROG terminates, the system reverts to the console for console input.

```
A>GET FILE XIN2 [SYSTEM]
```

The preceding command immediately directs the system to get subsequent console input from file XIN2 because it includes the SYSTEM option. The system reverts to the console for console input when it reaches the end of file in XIN2. Or, XIN2 can redirect the system back to the console if it contains a GET CONSOLE command.

## Terminate Console Input from a File

**Syntax:** GET {CONSOLE INPUT FROM} CONSOLE

**Explanation:** The preceding form of the GET command tells the system to get console input from the console.

**Example:**

```
A>GET CONSOLE
```

The preceding GET command tells the system to get console input from the console. You can use this command in a file (previously specified in a GET FILE command) which is already being read by the system for console input. It is used to redirect the console input to the console before the end of the file is reached.

# The HELP Command

**Syntax:** HELP {topic}{subtopic1 subtopic2...subtopic8}{[NO PAGE|LIST]}
HELP [EXTRACT]
HELP [CREATE]

**Explanation:** The HELP command is a transient utility that provides summarized information for all of the CP/M 3 commands described in this manual. In the distributed CP/M 3 system, HELP presents general information on a command as a topic and detailed information on a command as a subtopic. HELP with no command tail displays a list of all the available topics. HELP with a topic in the command tail displays information about that topic, followed by any available subtopics. HELP with a topic and a subtopic displays information about the specific subtopic.

After HELP displays the information for your specified topic, it displays the special prompt HELP> on your screen. Subtopics can be accessed by preceding the subtopic with a period. The period causes the subtopic search to begin at the last known level. You can continue to specify topics for additional information, or simply press the RETURN key to return to the CP/M 3 system prompt.

You can abbreviate the names of topics and subtopics. Usually one or two letters is enough to specifically identify the topics.

## Display Information

**Syntax:** HELP topic {subtopic1...subtopic8}{[NO PAGE|LIST]}
HELP>.Subtopic

**Explanation:** The preceding forms of the HELP command display the information for the specified topic and subtopics. Use the following two options with this form of the HELP command:

- The NOPAGE option disables the default paged display of every n lines, where n is the number of lines per page as set by the system or as set by the user. To stop the display, press CTRL-S. To resume the display, press CTRL-Q. You can abbreviate NOPAGE to N. (See the DEVICE command for more information about setting the number of lines per page.)

- The LIST option is the same as NOPAGE, except that it eliminates extra lines between headings. Use this option with CTRL-P to list the help information on the printer.

**Examples:**

```
A>HELP
```

The preceding command displays a list of topics for which help is available.

```
A>HELP DATE
```

This command displays general information about the DATE command. It also displays any available subtopics.

```
A>HELP DIR OPTIONS [N]
```

The preceding command includes the subtopic options. In response, HELP displays information about options associated with the DIR command. The display is not in paged mode.

```
A>HELP ED
```

The preceding command displays general information about the ED utility.

```
A>HELP ED COMMANDS
```

This form of HELP displays information about commands internal to ED. The preceding example can also be entered as

```
A>HELP ED
HELP>.COMMANDS
```

## Add Your Own Descriptions to the HELP.HLP File

**Syntax:** HELP [EXTRACT]
HELP [CREATE]

**Explanation:** CP/M 3 is distributed with two related HELP files: HELP.COM and HELP.HLP. The HELP.COM file is the command file that processes the text of the HELP.HLP file and displays it on the screen. The HELP.HLP file is a text file to which you can add customized information, but you cannot directly edit the HELP.HLP file. You must use the HELP.COM file to convert HELP.HLP to a file named HELP.DAT before you can edit or add your own text.

This form of the HELP command has the following options:

- The EXTRACT option accesses the file HELP.HLP on the default drive and creates a file called HELP.DAT on the default drive. You can now invoke a word processing program to edit or add your own text to the HELP.DAT file. EXTRACT can be abbreviated to E.
- The CREATE option accesses your edited HELP.DAT file on the default drive and builds a revised HELP.HLP file on the default drive. CREATE can be abbreviated to C.

You must add topics and subtopics to the HELP.DAT file in a specific format. A topic heading in the HELP.DAT file takes the form:

///nTopicname<cr>

The three backslashes are the topic delimiters and must begin in column one. In the preceding format statement, n is a number in the range from 1 through 9 that signifies the level of the topic. A main topic always has a level number of 1. The first subtopic has a level number of 2. The next level of subtopic has a level number of 3, and so forth, up to a maximum of nine levels. Topicname is the name of your topic, and allows a maximum of twelve characters. The entire line is terminated with a carriage return.

Use the following guidelines to edit and insert text into the HELP.DAT file.

- Topics should be placed in alphabetical order.
- Subtopics should be placed alphabetically within their respective supertopic.
- Levels must be indicated by a number 1-9.

Some examples of topic and subtopic lines in the HELP.HLP file follow

///1NEW UTILITY<cr>

///2COMMANDS<cr>

///3PARAMETERS<cr>

///2EXAMPLES<cr>

The first example illustrates the format of a main topic line. The second example shows how to number the first subtopic of that main topic. The third example shows how the next level subtopic under level 2 should be numbered. The fourth example shows how to return to the lower level subtopic. Any topic name with a level number of 1 is a main topic. Any topic name with a level number of 2 is a subtopic within its main topic.

When you are executing the HELP.COM file, you need only enter enough letters of the topic to unambiguously identify the topic name. When referencing a subtopic, you must type the topic name AND the subtopic, otherwise the HELP program cannot determine which main topic you are referencing. You can also enter a topic and subtopic following the program's internal prompt, HELP>, as follows

```
HELP>ED COMMANDS
```

This form of HELP displays information about commands internal to the editing program, ED.

# The HEXCOM Command

**Syntax:** HEXCOM filename

**Explanation:** The HEXCOM command is a transient utility that generates a command file (filetype COM) from a HEX input file. It names the output file with the same filename as the input file but with filetype COM. HEXCOM always looks for a file with filetype HEX.

**Example:**

```
A>HEXCOM B:PROGRAM
```

In the preceding command, HEXCOM generates a command file PROGRAM.COM from the input hex file PROGRAM.HEX.

# The INITDIR Command

**Syntax:** INITDIR d:

**Explanation:** The INITDIR command can initialize a disk directory to allow date and time stamping of files on that disk or remove date and time stamps.

You must use INITDIR to initialize the directory for any disk on which you plan to record date and time stamps for your files. If the disk is blank, INITDIR initializes the directory to record date and time stamps. If files already exist on the disk, INITDIR checks the space available for date and time stamps in the directory. If there is not enough room for date and time stamps, INITDIR does not initialize the directory and returns an error message.

After you initialize the directory for date and time stamps, you must use the SET command to specify time stamp options on the disk.

**Examples:** `A>INITDIR C:`

The system prompts to confirm:

```
INITDIR WILL ACTIVATE TIME STAMPS FOR SPECIFIED DRIVE.
Do you really want to re-format the directory: C (Y/N)?
```

If the directory has previously been initialized for date and time stamps, INITDIR displays the message:

```
Directory already re-formatted
Do you wish to recover date/time directory space
(Y/N)?
```

Enter Y to reinitialize the directory to eliminate date and time stamps. If you enter N, date and time stamping remains active on your disk and INITDIR displays the following message:

```
Do you want the existing date/time stamps cleared (Y/N)?
```

Enter Y to clear the existing stamps. Enter N to keep the existing date and time stamps.

# The LIB Command

**Syntax:** LIB filespec{[I|M|P|D]}
LIB filespec{[I|M|P]} = filespec{modifier}
{,filespec{modifier} ... }

**Explanation:** A library file contains a collection of object modules. Use the LIB utility to create libraries, and to append, replace, select, or delete modules from an existing library. You can also use LIB to obtain information about the contents of library files.

LIB creates and maintains library files that contain object modules in MicroSoft® REL format. These modules are produced by Digital Research's relocatable macro-assembler program, RMAC, or any other language translator that produces modules in MicroSoft REL format.

LINK-80™ links the object modules contained in a library to other object files. LINK-80 automatically selects from the library only those modules needed by the program being linked, and then forms an executable file with a filetype of COM.

The library file has the filetype REL or IRL depending on the option you choose. Modules in a REL library file must not contain backward references to modules that occur earlier in the library, because LINK-80 currently makes only one pass through a library.

**Table 5-8. LIB Options**

| *Option* | *Meaning* |
|---|---|
| I | The INDEX option creates an indexed library file of type IRL. LINK-80 searches faster on indexed libraries than on nonindexed libraries. |
| M | The MODULE option displays module names. |
| P | The PUBLICS option displays module names and the public variables for the new library file. |
| D | The DUMP option displays the contents of object modules in ASCII form. |

Use modifiers in the command line to instruct LIB to delete, replace, or select modules in a library file. Angle brackets enclose the modules to be deleted or replaced. Parentheses enclose the modules to be selected.

Unless otherwise specified, LIB assumes a filetype of REL for all source filenames. When you follow a filename by a group of module names enclosed in parentheses, these modules are included in the new library file. If modules are not specified, LIB includes all modules from the source file in the new library file.

**Table 5-9. LIB Modifiers**

| *Modifier* | *Meaning* |
|---|---|
| Delete | <module=> |
| Replace | <module=filename.REL><br><br>If module name and filename are the same this shorthand can be used:<br><br><filename> |
| Select | (modFIRST-modLAST,mod1,mod2,...,modN) |

**Examples:**

```
A>LIB TEST4[P]
A>LIB TEST5[P]=FILE1,FILE2
```

The first example displays all modules and publics in TEST4.REL. The second example creates TEST5.REL from FILE1.REL and FILE2.REL, and displays all modules and publics in TEST5.REL.

```
A>LIB TEST=TEST1(MOD1,MOD4),TEST2(C1-C4,C6)
```

In the preceding example LIB creates a library file TEST.REL from modules in two source files. TEST1.REL contributes MOD1 and MOD4. LIB extracts modules C1, C4, all the modules located between them, and module C6 from TEST2.REL.

```
A>LIB FILE2=FILE3<MODA=>
```

In this example, LIB creates FILE2.REL from FILE3.REL, omitting MODA which is a module in FILE3.REL.

```
A>LIB FILE6=FILE5<MODA=FILEB.REL>
A>LIB FILE6=FILE5<THISNAME>
```

In the first example, MODA is in the existing FILE5.REL. When LIB creates FILE6.REL from FILE5.REL, FILEB.REL replaces MODA.

In the second example, module THISNAME is in FILE5.REL. When LIB creates FILE6.REL from FILE5.REL the file THISNAME.REL replaces the similarly named module THISNAME.

```
A>LIB FILE1[I]=B:FILE2(PLOTS,FIND,SEARCH-DISPLAY)
```

In this example LIB creates FILE1.IRL on drive A from the selected modules PLOTS, FIND, and modules SEARCH through the module DISPLAY, in FILE2.REL on drive B.

# The LINK Command

**Syntax:** LINK d:{filespec,{[o]} = }filespec{[o]}{,...}

**Explanation:** The LINK command combines relocatable object modules such as those produced by RMAC and PL/I-80™ into a .COM file ready for execution. Relocatable files can contain external references and publics. Relocatable files can reference modules in library files. LINK searches the library files and includes the referenced modules in the output file. The LINK command is the LINK-80 utility and are synonymous in this discussion. See the *Programmer's Utilities Guide for the CP/M Family of Operating Systems* for a complete description of LINK-80.

You can use LINK option switches to control the execution parameters of LINK-80. LINK options follow the file specifications and are enclosed within square brackets. Multiple switches are separated by commas.

**Table 5-10. LINK Options**

| *Option* | *Meaning* |
|---|---|
| A | Additional memory; reduces buffer space and writes temporary data to disk. |
| B | BIOS link in banked CP/M 3 system. Aligns data segment on page boundary; puts length of code segment in header; defaults to SPR filetype. |
| Dhhhh | Data origin; sets memory origin for common and data area. |
| Gn | Go; set start address to label n. |
| Lhhhh | Load; change default load address of module to hhhh. Default 0100H. |
| Mhhhh | Memory size; define free memory requirements for MP/M™ modules. |
| NL | No listing of symbol table at console. |

Table 5-10. (continued)

| *Option* | *Meaning* |
|---|---|
| NR | No symbol table file. |
| OC | Output COM command file. Default. |
| OP | Output PRL page relocatable file for execution under MP/M in relocatable segment. |
| OR | Output RSP Resident System Process file for execution under MP/M. |
| OS | Output SPR System Page Relocatable file for execution under MP/M. |
| Phhhh | Program origin; changes default program origin address to hhhh. Default is 0100H. |
| Q | Lists symbols with leading question mark. |
| S | Search preceding file as a library. |
| $Cd | Destination of console messages, d, can be X for console, Y for printer, or Z for zero output. Default is X. |
| $Id | Source of intermediate files; d is disk drive A-P. Default is current drive. |
| $Ld | Source of library files; d is disk drive A-P. Default is current drive. |
| $Od | Destination of object file; d can be Z, or disk drive A-P. Default is to same drive as first file in the LINK-80 command. |
| $Sd | Destination of symbol file; d can be Y, Z, or disk drive A-P. Default is to same drive as first file in LINK-80 command. |

**Examples:** `A>LINK b:MYFILE[NR]`

LINK-80 on drive A uses as input MYFILE.REL on drive B and produces the executable machine code file MYFILE.COM on drive B. The [NR] option specifies no symbol table file.

`A>LINK m1,m2,m3`

LINK-80 combines the separately compiled files m1, m2, and m3, resolves their external references, and produces the executable machine code file m1.COM.

`A>LINK m=m1,m2,m3`

LINK-80 combines the separately compiled files m1, m2, and m3 and produces the executable machine code file m.COM.

`A>LINK MYFILE,FILE5[s]`

The [s] option tells LINK-80 to search FILE5 as a library. LINK-80 combines MYFILE.REL with the referenced subroutines contained in FILE5.REL on the default drive A and produces MYFILE.COM on drive A.

# The MAC Command

**Syntax:** MAC filename {$options}

**Explanation:** MAC, the CP/M Macro Assembler, is a transient utility that reads assembly language statements from a disk file of filetype ASM. MAC assembles the statements and produces three output files with the input filename and output filetypes of HEX, PRN, and SYM.

Filename.HEX contains Intel® hexadecimal format object code. You can debug the HEX file with a debugger, or use HEX COM to create a COM file and execute it.

Filename.PRN contains an annotated source listing that can be printed or examined at the console. The PRN file includes a 16-column wide listing at the left side of the page that shows the values of literals, machine code addresses, and generated machine code. An equal sign denotes literal addresses to eliminate confusion with machine code addresses.

Filename.SYM contains a sorted list of symbols defined in the program.

Before invoking MAC, you must prepare a source program file with the filetype ASM containing assembly language statements.

You can direct the input and output of MAC using the options listed in the following table. Use a letter with the option to indicate the source and destination drives, console, printer, or zero output. Valid drive names are A through O. X directs output to the console. P directs output to the printer. Z specifies that output files will not be created.

Table 5-11. Input/Output Options

| *Option* | *Meaning* |
|---|---|
| A | source drive for ASM file (A-O) |
| H | destination drive for HEX file (A-O, Z) |
| L | source drive for macro library LIB files called by the MACLIB statement. |
| P | destination drive for PRN file (A-O, X, P, Z) |
| S | destination drive for SYM file (A-O, X, P, Z) |

Table 5-12. Output File Modifiers

| *Modifier* | *Meaning* |
|---|---|
| +L | lists input lines read from macro library LIB files |
| −L | suppresses listing (default) |
| +M | lists all macro lines as they are processed during assembly |
| −M | suppresses all macro lines as they are read during assembly |
| *M | lists only hex generated by macro expansions |
| +Q | lists all LOCAL symbols in the symbol list |
| −Q | suppresses all LOCAL symbols in the symbol list (default) |
| +S | appends symbol file to print file |
| −S | suppresses creation of symbol file |
| +1 | produces a pass 1 listing for macro debugging in PRN file |
| −1 | suppresses listing on pass 1 (default) |

**Examples:**

```
A>MAC SAMPLE
```

In the preceding example MAC is invoked from drive A and operates on the file SAMPLE.ASM also on drive A.

```
A>MAC SAMPLE $PB AA HB SX
```

In this example, an assembly option parameter list follows the MAC command and the source filename. The parameters direct the PRN file to drive B, obtain the ASM file from drive A, direct the HEX file to drive B, and send the SYM file to the console. You can use blanks between option parameters.

# The PATCH Command

**Syntax:** PATCH filename {typ} {n}

**Explanation:** The PATCH command displays or installs patch number n to the CP/M 3 system or CP/M 3 command files.

Only CP/M 3 system files of filetype COM, PRL, or SPR can be patched with the PATCH command. If the typ option is not specified, the PATCH utility looks for a file with a filetype of COM.

The patch number n must be between 1 and 32 inclusive.

**Examples:**

```
A>PATCH SHOW 2
```

The preceding command patches the system SHOW.COM file with patch number 2. The system displays the following question:

```
Do you want to indicate that Patch #2
has been installed for SHOW.COM?Y
```

If the patch is successful, the system displays the message:

```
Patch Installed
```

If the patch is not successful, the system displays the following message:

```
Patch not Installed
```

One of the following error messages might be displayed:

- ERROR: Patch requires CP/M 3.
- ERROR: Invalid filetype typ.
- ERROR: Serial Number mismatch.
- ERROR: Invalid patch number n

# The PIP Command

**Syntax:** PIP dest-filespec|d:{[Gn]} = src-filespec{[o]}{,...} | d: {[o]}

**Explanation:** PIP is a transient utility that copies one or more files from one disk and/or user number to another. PIP can rename a file after copying it. PIP can combine two or more files into one file. PIP can also copy a character file from disk to the printer or other auxiliary logical output device. PIP can create a file on disk from input from the console or other logical input device. PIP can transfer data from a logical input device to a logical output device, thus the name Peripheral Interchange Program.

PIP copies file attributes with the file. This includes Read-Write or Read-Only and SYS or DIR file attributes and the user-definable attributes F1 through F4. If a file is password-protected, you must enter the password in the command line following the filename and/or filetype to which it belongs. If the password fails, the file is skipped and the failure noted.

When you specify a destination file with a password, PIP assigns that password to the destination file and automatically sets the password protection mode to READ. When you specify a destination file with no password, PIP does not assign a password to the destination file. When you specify only a destination drive, PIP assigns the same password and password protection mode to the destination file as specified in the source file. When you specify a destination file with a password, PIP automatically sets the password protection mode to READ. This means that you need a password to read the file. (See the SET command.)

## Single File Copy

**Syntax:** PIP d:{[Gn]} = src-filespec{[options]}

PIP dest-filespec{[Gn]} = d:{[options]}

PIP dest-filespec{[Gn]} = src-filespec{[o]}

**Explanation:** The first form shows the simplest way to copy a file. PIP looks for the file named by src-filespec on the default or optionally specified drive. PIP copies the file to the drive specified by d: and gives it the name specified by src-filespec. If you want, you can use the [Gn] option to place your destination file (dest-filespec) in the user number specified by n. The only option recognized for the destination file is [Gn]. Several options can be combined together for the source file specification (src-filespec). See the Table 5-13, PIP options.

The second form is a variation of the first. PIP looks for the file named by dest-filespec on the drive specified by d:, copies it to the default or optionally specified drive, and gives it the name specified by dest-filespec.

The third form shows how to rename the file after you copy it. You can copy it to the same drive and user number, or to a different drive and/or user number. Rules for options are the same. PIP looks for the file specified by src-filespec, copies it to the location specified in dest-filespec, and gives it the name indicated by dest-filespec.

Remember that PIP always goes to and gets from the current default user number unless you specify otherwise with the [Gn] option.

Before you start PIP, be sure that you have enough free space in kilobytes on your destination disk to hold the entire file or files that you are copying. Even if you are replacing an old copy on the destination disk with a new copy, PIP still needs enough room for the new copy before it deletes the old copy. Use the DIR command to determine filesize and the SHOW command to determine disk space. If there is not enough space, you can delete the old copy first by using the ERASE command.

Data is first copied to a temporary file to ensure that the entire data file can be constructed in the space available on the disk. PIP gives the temporary file the filename specified for the destination, with the filetype $$$. If the copy operation is successful, PIP changes the temporary filetype $$$ to the filetype specified in the destination.

If the copy operation succeeds and a file with the same name as the destination file already exists, the old file with the same name is erased before renaming the temporary file.

File attributes (DIR, SYS, RO, RW) are transferred with the files.

If the existing destination file is set to Read-Only (RO), PIP asks you if you want to delete it. Answer Y or N. Use the [W] option to write over Read-Only files.

You can include PIP options following each source name. There is one valid option ([Gn]—go to user number n) for the destination file specification. Options are enclosed in square brackets. Several options can be included for the source files. They can be packed together or separated by spaces. Options can verify that a file was copied correctly, allow PIP to read a file with the system (SYS) attribute, cause PIP to write over Read-Only files, cause PIP to put a file into or copy it from a specified user number, transfer from lower- to upper-case, and much more.

**Examples:**

```
A>PIP B:=A:oldfile.dat
A>PIP B:oldfile.dat = A:
```

Both forms of this command cause PIP to read the file oldfile.dat from drive A and put an exact copy of it onto drive B. This is called the short form of PIP, because the source or destination names only a drive and does not include a filename. When using this form you cannot copy a file from one drive and user number to the same drive and user number. You must put the destination file on a different drive or in a different user number. (See the section on PIP Options, and the USER Command.) The second short form produces exactly the same result as the first one. PIP looks for the file oldfile.dat on drive A, the drive specified as the source.

```
A>PIP B:newfile.dat=A:oldfile.dat
```

This command copies the file oldfile.dat from drive A to drive B and renames it to newfile.dat. The file remains as oldfile.dat on drive A. This is the long form of the PIP command, because it names a file on both sides of the command line.

```
A>PIP newfile.dat = oldfile.dat
```

Using this long form of PIP, you can copy a file from one drive and user number (usually user 0 because CP/M 3 automatically starts out in user 0—the default user number) to the same drive and user number. This gives you two copies of the same file on one drive and user number, each with a different name.

```
A>PIP B:PROGRAM.BAK = A:PROGRAM.DAT[G1]
```

The preceding command copies the file PROGRAM.DAT from user 1 on drive A to the current selected user number on drive B and renames the filetype on drive B to BAK.

```
B>PIP program2.dat = A:program1.dat[E V G3]
```

In this command, PIP copies the file named program1.dat on drive A and echoes [E] the transfer to the console, verifies [V] that the two copies are exactly the same, and gets [G3] the file program1.dat from user 3 on drive A. Because there is no drive specified for the destination, PIP automatically copies the file to the default user number and drive, in this case user 0 and drive B.

## Multiple File Copy

**Syntax:** PIP d:{[Gn]} ={d:}wildcard-filespec{[options]}

**Explanation:** When you use a wildcard in the source specification, PIP copies matching files one-by-one to the destination drive, retaining the original name of each file. PIP displays the message COPYING followed by each filename as the copy operation proceeds. PIP issues an error message and aborts the copy operation if the destination drive and user number are the same as those specified in the source.

**Examples:**

```
A>PIP B:=A:*.COM
```

This command causes PIP to copy all the files on drive A with the filetype COM to drive B.

```
A>PIP B:=A:*.*
```

This command causes PIP to copy all the files on drive A to drive B. You can use this command to make a back-up copy of your distribution disk. Note, however, that this command does not copy the CP/M 3 system from the system tracks. COPYSYS copies the system for you.

```
A>PIP B:=A:PROG????.*
```

The preceding command copies all files whose filenames begin with PROG from drive A to drive B.

```
A>PIP B:[G1]=A:*.BAS
```

This command causes PIP to copy all the files with a filetype of BAS on drive A in the default user number (user 0) to drive B in user number 1. Remember that the DIR, TYPE, ERASE, and other commands only access files in the same user number from which they were invoked. (See the USER Command.)

## Combining Files

**Syntax:** PIP dest-filespec{[Gn]} = src-filespec{[o]}, src-filespec{[o]}{,...}

**Explanation:** This form of the PIP command lets you specify two or more files in the source. PIP copies the files specified in the source from left to right and combines them into one file with the name indicated by the destination file specification. This procedure is called file concatenation. You can use the [Gn] option after the destination file to place it in the user number specified by n. You can specify one or more options for each source file.

Some of the options force PIP to copy files character-by-character. In these cases, PIP looks for a CTRL-Z character to determine where the end of the file is. All of the PIP options force a character transfer except the following:

A, C, Gn, K, O, R, V, and W.

Copying data to or from logical devices also forces a character transfer.

You can terminate PIP operations by typing CTRL-C.

When concatenating files, PIP only searches the last record of a file for the CTRL-Z end-of-file character. However, if PIP is doing a character transfer, it stops when it encounters a CTRL-Z character.

Use the [O] option if you are concatenating machine code files. The [O] option causes PIP to ignore embedded CTRL-Z (end-of-file) characters, which indicate the end-of-file character in text files, but might be valid data in object code files.

**Examples:**

```
A>PIP NEWFILE=FILE1,FILE2,FILE3
```

The three files named FILE1, FILE2, and FILE3 are joined from left to right and copied to NEWFILE.$$$. NEWFILE.$$$ is renamed to NEWFILE upon successful completion of the copy operation. All source and destination files are on the disk in the default drive A.

```
A>PIP B:X.BAS = Y.BAS, B:Z.BAS
```

The file Y.BAS on drive A is joined with Z.BAS from drive B and placed in the temporary file X.$$$ on drive B. The file X.$$$ is renamed to X.BAS on drive B when PIP runs to successful completion.

## Copy Files to and from Auxiliary Devices

**Syntax:**

```
PIP dest-filespec{[Gn]} = src-filespec{[o]}
    AUX:                  AUX:{[o]}
    CON:                  CON:{[o]}
    PRN:                  NUL:
    LST:                  EOF:
```

**Explanation:** This form is a special case of the PIP command line that lets you copy a file from a disk to a device, from a device to a disk or from one device to another. The files must contain printable characters. Each peripheral device is assigned to a logical device that identifies a source device that can transmit data or a destination device that can receive data. (See the DEVICE command.) A colon follows each logical device name so it cannot be confused with a filename. Enter CTRL-C to abort a copy operation that uses a logical device in the source or destination.

The logical device names are listed as follows:

- CON: Console input or output device. When used as a source, usually the keyboard; when used as a destination, usually the screen.
- AUX: Auxiliary Input or Output Device.
- LST: The destination device assigned to the list output device, usually the printer.

The following three device names have special meaning:

- NUL: A source device that produces 40 hexadecimal zeros.
- EOF: A source device that produces a single CTRL-Z, the CP/M 3 end-of-file mark.
- PRN: The printer device with tab expansion to every eighth column, line numbers, and page ejects every sixtieth line.

**Examples:**

```
B>PIP PRN:=CON:,MYDATA.DAT
```

Characters are first read from the console input device, generally the keyboard, and sent directly to your printer device. You type a CTRL-Z character to tell PIP that keyboard input is complete. At that time, PIP continues by reading character data from the file MYDATA.DAT on drive B. Because PRN: is the destination device, tabs are expanded, line numbers are added, and page ejects occur every sixty lines.

Note that when the CON: device is the source you must enter both the carriage return (RETURN) and line-feed (LF) keys for a new line.

```
A>PIP B:FUNFILE.SUE = CON:
```

Whatever you type at the console is written to the file FUNFILE.SUE on drive B. End the keyboard input by typing a CTRL-Z.

```
A>PIP LST:=CON:
```

Whatever you type at the console keyboard is written to the list device, generally the printer. Terminate input with a CTRL-Z.

```
A>PIP LST:=B:DRAFT.TXT[T8]
```

The file DRAFT.TXT on drive B is written to the printer device. Any tab characters are expanded to the nearest column that is a multiple of 8.

```
A>PIP PRN:=B:DRAFT.TXT
```

The preceding command causes PIP to write the file DRAFT.TXT to the list device. It automatically expands the tabs, adds line numbers, and ejects pages after sixty lines.

## Multiple Command Mode

**Syntax:** PIP

**Explanation:** This form of the PIP command starts the PIP utility and lets you type multiple command lines while PIP remains in user memory.

PIP writes an asterisk on your screen when ready to accept input command lines.

You can type any valid command line described under previous PIP formats following the asterisk prompt.

Terminate PIP by pressing only the RETURN key following the asterisk prompt. The empty command line tells PIP to discontinue operation and return to the CP/M 3 system prompt.

**Examples:**

```
A>PIP
CP/M 3 PIP VERSION 3.0
*NEWFILE=FILE1,FILE2,FILE3
* APROG.COM=BPROG.COM
* A:=B:X.BAS
*B:=*.*
*^M
A>
```

This command loads the PIP program. The PIP command input prompt, *, tells you that PIP is ready to accept commands. The effects of this sequence of commands are the same as in the previous examples, where

the command line is included in the command tail. PIP is not loaded into memory for each command. To exit this PIP command mode, press RETURN or one of its equivalent control characters, CTRL-J or CTRL-M as shown.

## Using Options With PIP

**Explanation:** With options you can process your source file in special ways. You can expand tab characters, translate from upper- to lower-case, extract portions of your text, verify that the copy is correct, and much more.

The PIP options are listed in Table 5-13 using n to represent a number and s to represent a sequence of characters terminated by a CTRL-Z. An option must immediately follow the file or device it affects. The option must be enclosed in square brackets [ ]. For those options that require a numeric value, no blanks can occur between the letter and the value.

You can include the [Gn] option after a destination file specification. You can include a list of options after a source file or source device. An option list is a sequence of single letters and numeric values that are optionally separated by blanks and enclosed in square brackets [ ].

Table 5-13. PIP Options

| *Option* | *Function* |
|---|---|
| A | Copy only the files that have been modified since the last copy. To back up only the files that have been modified since the last back-up, use PIP with the archive option, [A]. |
| C | Prompt for confirmation before performing each copy operation. Use the [C] option when you want to copy only some files of a particular filetype. |
| Dn | Delete any characters past column n. This parameter follows a source file that contains lines too long to be handled by the destination device, for example, an 80-character printer or narrow console. The number n should be the maximum column width of the destination device. |
| E | Echo transfer at console. When this parameter follows a source name, PIP displays the source data at the console as the copy is taking place. The source must contain character data. |
| F | Filter form-feeds. When this parameter follows a source name, PIP removes all form-feeds embedded in the source data. To change form-feeds set for one page length in the source file to another page length in the destination file, use the F command to delete the old form-feeds and a P command to simultaneously add new form-feeds to the destination file. |
| Gn | Get source from or go to user number n. When this parameter follows a source name, PIP searches the directory of user number n for the source file. When it follows the destination name, PIP places the destination file in the user number specified by n. The number must be in the range 0 to 15. |

**Table 5-13. (continued)**

| *Option* | *Function* |
| --- | --- |
| H | Hex data transfer. PIP checks all data for proper Intel hexadecimal file format. The console displays error messages when errors occur. |
| I | Ignore :00 records in the transfer of Intel hexadecimal format file. The I option automatically sets the H option. |
| L | Translate upper-case alphabetics in the source file to lower-case in the destination file. This parameter follows the source device or filename. |
| N | Add line numbers to the destination file. When this parameter follows the source filename, PIP adds a line number to each line copied, starting with 1 and incrementing by one. A colon follows the line number. If N2 is specified, PIP adds leading zeros to the line number and inserts a tab after the number. If the T parameter is also set, PIP expands the tab. |
| O | Object file transfer for machine code (noncharacter and therefore nonprintable) files. PIP ignores any CTRL-Z end-of-file during concatenation and transfer. Use this option if you are combining object code files. |
| Pn | Set page length. n specifies the number of lines per page. When this parameter modifies a source file, PIP includes a page eject at the beginning of the destination file and at every n lines. If n = 1 or is not specified, PIP inserts page ejects every sixty lines. When you also specify the F option, PIP ignores form-feeds in the source data and inserts new form-feeds in the destination data at the page length specified by n. |

Table 5-13. (continued)

| *Option* | *Function* |
|---|---|
| Qs | Quit copying from the source device after the string s. When used with the S parameter, this parameter can extract a portion of a source file. The string argument must be terminated by CTRL-Z. |
| R | Read system (SYS) files. Usually, PIP ignores files marked with the system attribute in the disk directory. But when this parameter follows a source filename, PIP copies system files, including their attributes, to the destination. |
| Ss | Start copying from the source device at the string s. The string argument must be terminated by CTRL-Z. When used with the Q parameter, this parameter can extract a portion of a source file. Both start and quit strings are included in the destination file. |
| Tn | Expand tabs. When this parameter follows a source filename, PIP expands tab (CTRL-I) characters in the destination file. PIP replaces each CTRL-I with enough spaces to position the next character in a column divisible by n. |
| U | Translate lower-case alphabetic characters in the source file to upper-case in the destination file. This parameter follows the source device or filename. |
| V | Verify that data has been copied correctly. PIP compares the destination to the source data to ensure that the data has been written correctly. The destination must be a disk file. |

Table 5-13. (continued)

| *Option* | *Function* |
|---|---|
| W | Write over files with RO (Read-Only) attribute. Usually, if a PIP command tail includes an existing RO file as a destination, PIP sends a query to the console to make sure you want to write over the existing file. When this parameter follows a source name, PIP overwrites the RO file without a console exchange. If the command tail contains multiple source files, this parameter need follow only the last file in the list. |
| Z | Zero the parity bit. When this parameter follows a source name, PIP sets the parity bit of each data byte in the destination file to zero. The source must contain character data. |

**Examples:**

```
A>PIP NEWPROG.BAS=CODE.BAS[L], DATA.BAS[U]
```

This command constructs the file NEWPROG.BAS on drive A by joining the two files CODE.BAS and DATA.BAS from drive A. During the copy operation, CODE.BAS is translated to lower-case, while DATA.BAS is translated to upper-case.

```
A> PIP CON:=WIDEFILE.BAS[D80]
```

This command writes the character file WIDEFILE.BAS from drive A to the console device, but deletes all characters following the 80th column position.

```
A>PIP B:=LETTER.TXT[E]
```

The file LETTER.TXT from drive A is copied to LETTER.TXT on drive B. The LETTER.TXT file is also written to the screen as the copy operation proceeds.

```
A>PIP LST:=B:LONGPAGE.TXT[FP65]
```

This command writes the file LONGPAGE.TXT from drive B to the printer device. As the file is written, form-feed characters are removed and reinserted at the beginning and every 65th line thereafter.

```
B>PIP LST:=PROGRAM.BAS[NT8U]
```

This command writes the file PROGRAM.BAS from drive B to the printer device. The N parameter tells PIP to number each line. The T8 parameter expands tabs to every eighth column. The U parameter translates lower-case letters to upper-case as the file is printed.

```
A>PIP PORTION.TXT=LETTER.TXT[SDear Sir^Z QSincerely^Z]
```

This command abstracts a portion of the LETTER.TXT file from drive A by searching for the character sequence "Dear Sir" before starting the copy operation. When found, the characters are copied to PORTION.TXT on drive A until the sequence "Sincerely" is found in the source file.

```
B>PIP B:=A:*.COM[VWR]
```

This command copies all files with filetype COM from drive A to drive B. The V parameter tells PIP to read the destination files to ensure that data was correctly transferred. The W parameter lets PIP overwrite any destination files that are marked as RO (Read-Only). The R parameter tells PIP to read files from drive A that are marked with the SYS (System) attribute.

# The PUT Command

**Syntax:**

PUT CONSOLE {OUTPUT TO} FILE filespec {[o]}
PUT PRINTER {OUTPUT TO} FILE filespec {[o]}
PUT CONSOLE {OUTPUT TO} CONSOLE
PUT PRINTER {OUTPUT TO} PRINTER

**Explanation:** The PUT command is a transient utility that lets you direct console output or printer output to a file. PUT allows you to direct the system to put console output or printer output to a file for the next system command or user program entered at the console. Or, PUT directs all subsequent console or printer output to a file when you include the SYSTEM option.

Console output is directed to a file until the program terminates. Then, console output reverts to the console. Printer output is directed to a file until the program terminates. Then printer output is directed back to the printer.

When you use the SYSTEM option, all subsequent console/printer output is directed to the specified file. This option terminates when you enter the PUT CONSOLE or PUT PRINTER command.

The syntax for the option list is

[{ECHO | NO ECHO} {FILTER | NO FILTER} | {SYSTEM}]

Table 5-14 defines the preceding option list.

Table 5-14. PUT Options

| *Option* | *Meaning* |
|---|---|
| ECHO | specifies that the output is echoed to the console. ECHO is the default option when you direct console output to a file. |
| NO ECHO | specifies that the file output is not to be echoed to the console. |
| FILTER | specifies that filtering of control characters is allowed, which means that control characters are translated to printable characters. For example, an escape character is translated to ^[. |
| NO FILTER | means that PUT does not translate control characters. This is the default option. |
| SYSTEM | specifies that system output and program output is written to the file specified by filespec. Output is written to the file until a subsequent PUT CONSOLE command redirects console output back to the console. |

## Direct Console Output to a File

**Syntax:** PUT CONSOLE {OUTPUT} TO FILE filespec {[o]}

**Explanation:** The preceding form of the PUT command tells the system to direct subsequent console output to a file.

**Example:** `A>PUT CONSOLE OUTPUT TO FILE XOUT [ECHO]`

The preceding command directs console output to file XOUT with the output echoed to the console.

## Put Printer Output to a File

**Syntax:** PUT PRINTER {OUTPUT TO} FILE filespec {[o]}

**Explanation:** The preceding form of the PUT command directs printer output to a file.

The options are the same as in the PUT CONSOLE command, except that option NO ECHO is the default for the PUT PRINTER command. Note that if ECHO is specified, printer output is echoed to the printer.

**Examples:**

```
A>PUT PRINTER OUTPUT TO FILE XOUT
A>MYPROG
```

The preceding example directs the printer output of program MYPROG to file XOUT. The output is not echoed to the printer.

```
A>PUT PRINTER OUTPUT TO FILE XOUT2 [ECHO,SYSTEM]
```

The preceding command directs all printer output to file XOUT2 and to the printer, and the PUT is in effect until you enter a PUT PRINTER OUTPUT TO PRINTER command.

The printer output can be directed to one or more files. The output to these files is terminated when you revert printer output to the printer using the following command:

PUT PRINTER OUTPUT TO PRINTER

## Terminate Console Output to a File

**Syntax:** PUT CONSOLE {OUTPUT TO} CONSOLE

**Explanation:** The preceding form of the PUT command directs console output to the console.

**Example:**

```
A>PUT CONSOLE OUTPUT TO CONSOLE
```

The preceding command directs console output to the console.

## Terminate Printer Output to a File

**Syntax:** PUT PRINTER {OUTPUT TO} PRINTER

**Explanation:** The preceding form of the PUT command directs the printer output to the printer.

**Example:**

```
A>PUT PRINTER OUTPUT TO PRINTER
```

The preceding example directs printer output to the printer.

# The RENAME Command

**Syntax:** RENAME {new-filespec = old-filespec}

**Explanation:** The RENAME command lets you change the name of a file that is cataloged in the directory of a disk. It also lets you change several filenames if you use wildcards in the filespecs. You can abbreviate RENAME to REN.

The new-filespec must not be the name of any existing file on the disk. The old-filespec identifies an existing file or files on the disk.

The RENAME command changes the file named by old-filespec to the name given as new-filespec.

RENAME does not make a copy of the file. RENAME changes only the name of the file.

If you omit the drive specifier, RENAME assumes the file to rename is on the default drive. You can include a drive specifier as a part of the newname. If both file specifications name a drive, it must be the same drive.

If the file given by oldname does not exist, RENAME displays the following message on the screen:

```
No File
```

If the file given by newname is already present in the directory, RENAME displays the following message on the screen:

```
Not renamed: filename.typ file already exists,
delete (Y/N)?
```

If you want to delete the old file, type Y to delete. Otherwise, type N to keep the old file and not rename the new file.

If you use wildcards in the filespecs, the wildcards in the new filespec must correspond exactly to the wildcards in the old filespec. For example, in the following two commands, the wildcard filespecs correspond exactly:

```
A>REN *.TX1=*.TEX
A>REN A*.T*=S*.T*
```

In the following example, the wildcards do not match and CP/M 3 returns an error message.

```
A>REN A*.TEX=A.T*
```

**Examples:**

```
A>RENAME NEWASM.BAS=OLDFILE.BAS
```

The file OLDFILE.BAS changes to NEWASM.BAS on drive A.

```
A>RENAME
```

The system prompts for the filespecs:

```
Enter New Name:X.PRN
Enter Old Name:Y.PRN
Y       .PRN=X         .PRN
A>
```

File Y.PRN is renamed X.PRN on drive A.

```
B>REN A:X.PAS = Y.PLI
```

The file Y.PLI changes to X.PAS on drive A.

```
A>RENAME S*.TEX=A*.TEX
```

The preceding command renames all the files matching the wildcard A*.TEX to files with filenames matching the wildcard S*.TEX, respectively.

```
A>REN B:NEWLIST=B:OLDLIST
```

The file OLDLIST changes to NEWLIST on drive B. Because the second drive specifier, B: is implied by the first, it is unnecessary in this example. The preceding command line has the same effect as the following:

```
A>REN B:NEWLIST=OLDLIST
```

or

```
A>REN NEWLIST=B:OLDLIST
```

# The RMAC Command

**Syntax:** RMAC filespec {$Rd | $Sd | $Pd}

**Explanation:** RMAC is a relocatable macro assembler that assembles files of type ASM into REL files that can be linked to create COM files.

The RMAC command options specify the destination of the output files. The additional specifier d defines the destination drive of the output files. A-O specifies drives A through O. X means output to the console, P means output to the printer, and Z means zero output. Table 5-15 lists the RMAC command options.

**Table 5-15. RMAC Command Options**

| *Option* | *d=output option* |
|---|---|
| R drive for REL file | (A-O, Z) |
| S drive for SYM file | (A-O, X, P, Z) |
| P drive for PRN file | (A-O, X, P, Z) |

In the MAC command, the assembly parameter of H controls the destination of the HEX file. In the RMAC command this parameter is replaced by R, which controls the destination of the REL file; however, you cannot direct the REL file to the console or printer, RX or RP, because the REL file is not an ASCII file.

**Examples:**

```
A>RMAC TEST $PX SB RB
```

In the preceding example RMAC assembles the file TEST.ASM from drive A, sends the listing file (TEST.PRN) to the console, puts the symbol file (TEST.SYM) on drive B and puts the relocatable object file (TEST.REL) on drive B.

# The SAVE Command

**Syntax:** SAVE

**Explanation:** The SAVE command copies the contents of memory to a file. To use the SAVE utility, first issue the SAVE command, then run your program which reads a file into memory. When your program exits, it exits to the SAVE utility. The SAVE utility prompts you for the filespec to which the memory is to be copied, and the beginning and ending address of the memory to be saved.

**Example:**

```
A>SAVE
```

The preceding command activates the SAVE utility. Now enter the name of the program that loads a file into memory.

```
A>SID dump.com
```

Next, execute the program.

```
#g0
```

When the program exits, SAVE intercepts the return to the system and prompts you for the filespec and the bounds of memory to be saved.

```
SAVE Ver 3.0
File (or RETURN to exit)?dump2.com
Delete dump2.com?Y
From?100
To?400

A>
```

The contents of memory from 100H, hexadecimal, to 400H is copied to file DUMP2.COM.

# The SET Command

**Syntax:** SET[options]
SET d: [options]
SET filespec [options]

**Explanation:** The SET command initiates password protection and time stamping of files in the CP/M 3 system. It also sets file and drive attributes, such as the Read-Only, SYS, and user-definable attributes. It lets you label a disk and password protect the label.

The SET command include options that affect the disk directory, the drive, or a file or set of files. The discussion of the SET command explicitly states which of the three categories are affected.

To enable time stamping of files, you must first run INITDIR to format the disk directory.

## Set File Attributes

**Syntax:** SET filespec[attribute-options]

**Explanation:** The preceding SET command sets the specified attributes of a file or a group of files.

**Table 5-16. SET File Attributes**

| *Option* | *Meaning* |
|---|---|
| DIR | Sets the file from the SYS directory to the (DIR) attribute. |
| SYS | Gives the file the System SYS attribute. |
| RO | Sets the file attribute to allow Read-Only access. |
| RW | Sets the file attribute to allow Read-Write access. |

Table 5-16. (continued)

| *Option* | *Meaning* |
|---|---|
| ARCHIVE=OFF | Sets the archive attribute to off. This means that the file has not been backed up (archived). PIP with the [A] option can copy files with the archive attribute set to OFF. PIP with this option requires an ambiguous filespec and copies only files that have been created or changed since the last time they were backed up with the PIP[A] option. PIP then sets the archive attribute to ON for each file successfully copied. |
| ARCHIVE=ON | Sets the archive attribute to on. This means that the file has been backed up (archived). The archive attribute can be turned on explicitly by the SET command, or it can be turned on by PIP when copying a group of files with the PIP [A] option. The archive attribute is displayed by DIR. |
| F1=ON\|OFF | Turns on or off the user-definable file attribute F1. |
| F2=ON\|OFF | Turns on or off the user-definable file attribute F2. |
| F3=ON\|OFF | Turns on or off the user-definable file attribute F3. |
| F4=ON\|OFF | Turns on or off the user-definable file attribute F4. |

**Example:** `A>SET MYFILE.TEX[RO SYS]`

The preceding command sets MYFILE.TEX to Read-Only and System.

```
A>SET MYFILE.TEX [RW DIR]
```

The preceding command sets MYFILE.TEX to Read-Write with the Directory (DIR) attribute.

## Set Drive Attribute

**Syntax:** SET {d:} [RO]
SET {d:} [RW]

**Explanation:** The preceding SET commands set the specified drive to Read-Only or Read-Write.

If a drive is set to Read-Only, PIP cannot copy a file to it, ERASE cannot delete a file from it, RENAME cannot rename a file on it. You cannot perform any operation that requires writing to the disk. When the specified drive is set to Read-Write, you can read or write to the disk in that drive. If you enter a CTRL-C at the system prompt, all drives are reset to Read-Write.

**Example:**

```
A>SET B:[RO]
```

The preceding command sets drive B to Read-Only.

## Assign a Label to the Disk

**Syntax:** SET {d:}[NAME=labelname.typ]

**Explanation:** The preceding SET command assigns a label (name) to the disk in the specified or default drive.

CP/M 3 provides a facility for creating a directory label for each disk. The directory label can be assigned an eight-character name and a three-character type similar to a filename and filetype. Label names make it easier to catalog disks and keep track of different disk directories. The default label name is LABEL.

**Example:** `A>SET[NAME=DISK100]`

The preceding example labels the disk on the default drive DISK100.

## Assign Password to the Label

**Syntax:** SET [PASSWORD=password]
SET [PASSWORD=<cr>

**Explanation:** The first form of the preceding SET command assigns a password to the disk label. The second form of the command removes password protection from the label.

You can assign a password to the label. If the label has no password, any user who has access to the SET program can set other attributes to the disk which might make the disk inaccessible to you. However, if you assign a password to the label, then you must supply the password to set any of the functions controlled by the label. SET always prompts for the password if the label is password-protected.

**Examples:**

```
A>SET[PASSWORD=SECRET]
A>SET [PASSWORD=<cr>
```

The first command assigns SECRET to the disk label. The second command nullifies the existing password.

**Note:** If you use password protection on your disk, be sure to record the password. If you forget the password, you lose access to your disk or files.

## Enable/Disable Password Protection for Files on a Disk

**Syntax:** SET [PROTECT=ON]
SET [PROTECT=OFF]

**Explanation:** The first form of the SET command turns on password protection for all the files on the disk. The password protection must be turned on before you can assign passwords to individual files or commands.

The second SET command disables password protection for the files on your disk.

After a password is assigned to the label and the PROTECT option is turned on, you are ready to assign passwords to your files.

You can always determine if a disk is password-protected by using the SHOW command to display the label.

## Assign Passwords to Files

**Syntax:** SET filespec[PASSWORD = password]

**Explanation:** The preceding SET command sets the password for filespec to the password indicated in the command tail. Passwords can be up to eight characters long. Lower-case letters are translated to upper-case.

You can use wildcards in the filespec. SET assigns the specified password to the files that match the wildcard-filespec.

**Note:** always record the passwords that you assign to your files. Without the password, you cannot access those files unless password protection is turned off for the whole disk. If you forget the password to the directory label, you cannot turn off the password protection for the disk.

**Example:**

```
A>SET MYFILE.TEX[PASSWORD=MYFIL]
```

MYFIL is the password assigned to file MYFILE.TEX.

## Set Password Protection Mode for Files with Passwords

**Syntax:**

SET filespec [PROTECT = READ]
SET filespec [PROTECT = WRITE]
SET filespec [PROTECT = DELETE]
SET filespec [PROTECT = NONE]

**Explanation:** You can assign one of four modes of password protection to your file. The protection modes are READ, WRITE, DELETE, and NONE and are described in the following table.

**Table 5-17. Password Protection Modes**

| *Mode* | *Protection* |
|---|---|
| READ | The password is required for reading, copying, writing, deleting, or renaming the file. |
| WRITE | The password is required for writing, deleting, or renaming the file. You do not need a password to read the file. |
| DELETE | The password is only required for deleting or renaming the file. You do not need a password to read or modify the file. |
| NONE | No password exists for the file. If a password exists, this modifier can be used to delete the password. |

## Assign a Default Password

**Syntax:** SET [DEFAULT=password]

**Explanation:** The preceding set command assigns a default password for the system to use during your computer session. The system uses the default password to access password-protected files if you do not specify a password, or if you enter an incorrect password. The system lets you access the file if the default password matches the password assigned to the file.

**Example:** `B>SET *.TEX[PASSWORD=SECRET, PROTECT=WRITE]`

The preceding command assigns the password SECRET to all the TEX files on drive B. Each TEX file is given a WRITE protect mode to prevent unauthorized editing.

**Example:** `A>SET[DEFAULT=dd]`

The preceding command instructs the system to use dd as a password if you do not enter a password for a password-protected file.

## Set Time Stamp Options on Disk

**Syntax:**
SET [CREATE=ON]
SET [ACCESS=ON]
SET [UPDATE=ON]

**Explanation:** The preceding SET commands allow you to keep a record of the time and date of file creation and update, or of the last access and update of your files.

| | |
|---|---|
| [CREATE=ON] | turns on CREATE time stamps on the disk in the default drive. To record the creation time of a file, the CREATE option must have been turned on before the file is created. |
| [ACCESS=ON] | turns on ACCESS time stamps on the disk in the default drive. ACCESS and CREATE options are mutually exclusive. This means that only one can be in effect at a time. If you turn on the ACCESS time stamp on a disk that has the CREATE time stamp, the CREATE time stamp is automatically turned off. |
| [UPDATE=ON] | turns on UPDATE time stamps on the disk in the default drive. UPDATE time stamps record the time the file was last modified. |

To enable time stamping, you must first run INITDIR to format the disk directory for time and date stamping.

Although there are three kinds of date/time stamps, only two date/time stamps can be associated with a given file at one time. You can choose to have either a CREATE date or an ACCESS date for files on a particular disk.

When you set both UPDATE and CREATE time stamps, notice that editing a file changes both the UPDATE and CREATE time stamps. This is because ED does not update the original file but creates a new version with the name of the original file.

**Example:** *A>SET[ACCESS=ON]*

The DIR with [FULL] option displays the following date and time stamps:

```
B>DIR [FULL]

Directory for Drive B:

    Name       Bytes  Recs  Attributes  Prot     Update          Access
------------   -----  ----  ----------  ----  --------------  --------------
ONE     .TEX      9k    71  Dir RW      None                  08/03/81 10:56
THREE   .TEX     12k    95  Dir RW      None                  08/05/81 15:45
TWO     .TEX     10k    76  Dir RW      None                  08/10/81 09:13
```

The access time stamps displayed show the time the file was last displayed or edited. Note that displaying a filename in a directory listing does not constitute an access and is not recorded.

*A>SET [CREATE=ON,UPDATE=ON]*

The following DIR output below shows how files with create and update time stamps are displayed.

```
B>DIR [FULL]

Directory for Drive B:

    Name       Bytes  Recs  Attributes  Prot     Update          Create
------------   -----  ----  ----------  ----  --------------  --------------
GENLED  .DAT    109k   873  Dir RW      None  08/05/81 14:01  08/01/81 09:36
RECEIPTS.DAT     59k   475  Dir RW      None  08/08/81 12:11  08/01/81 09:40
INVOICES.DAT     76k   608  Dir RW      None  08/08/81 08:46  08/01/81 10:15
```

## Additional SET Examples

**Examples:** `A>SET *.COM[SYS,RO,PASS=123,PROT=READ]`

The preceding setting gives the most protection for all the COM files on drive A. With the password protection mode set to READ, you cannot even read one of the COM files without entering the password 123, unless the default password has been set to 123. Even if the correct password is entered, you still cannot write to the file because the file is Read-Only.

`A>SET *.COM [RW,PROTECT=NONE,DIR]`

The preceding command reverses the protection and access attributes of the COM files affected by the previous example. After executing the preceding command, there is no password protection, the files of type COM can be read from or written to, and are set to DIR files.

# The SETDEF Command

**Syntax:** SETDEF {d:{,d:{,d:{,d:}}}} {TEMPORARY = d:] | [ORDER = (typ {,typ})]}
SETDEF [DISPLAY | NO DISPLAY]
SETDEF [PAGE | NO PAGE]

**Explanation:** The SETDEF command lets you display or define the disk search order, the temporary drive, and the filetype search order. The SETDEF definitions affect only the loading of programs and/or execution of SUBMIT (SUB) files. The SETDEF command also lets you turn on/off the DISPLAY and PAGE modes for the system. When DISPLAY mode is on, the system displays the location and name of programs loaded or SUB files executed. When PAGE mode is on, CP/M 3 utilities stop after displaying one full screen of information. Press any key to continue the display.

The system usually searches the specified drive or the default drive for files. The user can use the SETDEF command, to extend the search for program files and submit files, for execution purposes only.

**Note:** A CP/M 3 program file has a filetype of COM. A file containing commands to be executed by SUBMIT has a filetype of SUB.

## Display the Program Loading Search Definitions

**Syntax:** SETDEF

**Explanation:** The preceding form of the SETDEF command displays the disk search order, the temporary drive, and the filetype search order.

## Assign the Drive for Temporary Files

**Syntax:** SETDEF [TEMPORARY = D:]

**Explanation:** The preceding form of the SETDEF command defines the disk drive to be used for temporary files. The default drive used for temporary files is the system default drive.

**Example:** A>SETDEF [TEMPORARY=C:]

The preceding command sets disk drive C as the drive to be used for temporary files.

## Define the Disk Drive Search Order

**Syntax:** SETDEF { d: {,d: {,d: {,d:}}}}

**Explanation:** The preceding form of the SETDEF command defines the disks to be searched by the system for programs and/or submit files to be executed. The CP/M 3 default is to search only the default drive.

**Note:** * can be substituted for d: to indicate that the default drive is to be included in the drive search order.

**Example:** A>SETDEF C:,*

The preceding example tells the system to search for a program on drive C, then, if not found, search for it on the default drive.

## Define the Filetype Search Order

**Syntax:** SETDEF [ ORDER = (typ {,typ}) ]
where typ = COM or SUB

**Explanation:** The preceding form of the SETDEF command defines the filetype search order to be used by system for program loading. The filetype, indicated as typ in the syntax line, must be COM or SUB. The CP/M 3 default search is for COM files only.

**Example:** A>SETDEF [ORDER=(SUB,COM)]

The preceding command instructs the system to search for a SUB file to execute. If no SUB file is found, search for a COM file.

## Turn On/Off System Display Mode

**Syntax:** SETDEF [DISPLAY | NO DISPLAY]

**Explanation:** The preceding command turns the system display mode on or off. The default system display mode is off. When the display mode is on, CP/M 3 displays the following information about a program file before loading it for execution: drive, filename, filetype (if any), and user number (if not the default user number).

**Example:**

```
A>SETDEF [DISPLAY]
```

The preceding command turns on the system display mode. The system now displays the name and location of programs loaded or submit files executed. For example, if you enter the PIP command after turning on the system display mode, CP/M 3 displays the following:

```
A>PIP
A:PIP      COM
CP/M 3 PIP VERSION 3.0
*
```

indicating that the file PIP.COM was loaded from drive A under the current user number. If the current user number is not 0, and if PIP.COM does not exist under the current user number, then the system displays the location of PIP.COM as follows:

```
4A>PIP
A:PIP      COM (User 0)
CP/M 3 PIP VERSION 3.0
*
```

indicating that PIP.COM was loaded from drive A under user number 0. This mode is in effect until you enter

```
SETDEF [NO DISPLAY]
```

to turn off the system DISPLAY mode.

## Turn On/Off System Page Mode

**Syntax:** SETDEF [ PAGE| NO PAGE ]

**Explanation:** The preceding command turns on/off the system page mode. When the PAGE mode is set to on, CP/M 3 utilities stop after displaying one full screen of information, called a console page. The utilities resume after you press any key.

The default setting of the system page mode is ON.

**Example:** `A>SETDEF [NO PAGE]`

The preceding command turns off the system page mode. CP/M 3 utilities do not pause after displaying a full console page, but continue to scroll.

# The SHOW Command

**Syntax:** SHOW {d:}{[SPACE |LABEL |USERS |DIR |DRIVE]}

**Explanation:** The SHOW command displays the following disk drive information:

- access mode and amount of free disk space
- disk label
- current user number
- number of files for each user number on the disk
- number of free directory entries for the disk
- drive characteristics

## Display Access Mode and Disk Space Available

**Syntax:** SHOW {d:}{[SPACE]}

**Explanation:** The preceding form of the SHOW command displays the drive, the access mode for that drive, and the remaining space in kilobytes for the specified drive. SHOW by itself displays the information for all logged-in drives in the system.

**Examples:**

```
A>SHOW B:
B: RW, Space:    9,488k
A>SHOW
A: RW, Space:        4k
B: RW, Space:    9,488k
```

The first example shows that drive B has Read-Write access and it has 9,488K bytes of space left. The second example shows that drive A also is Read-Write and has only 4K bytes left and drive B is Read-Write and has 9,488K bytes left.

## Display Disk Label

**Syntax:** SHOW {d:}[LABEL]

**Explanation:** The preceding form of the SHOW command displays disk label information.

**Example:** *A>SHOW B:[LABEL]*

The preceding command displays the following for drive B:

```
Label for drive B:

Directory  Passwds   Stamp  Stamp  Label Created  Label Updated
Label      Reqd      Create Update
---------- -------   ------ ------ -------------- --------------
TOMSDISK.    on        on     on   07/04/81 10:30 07/08/81 09:30
```

The first column, directory label, displays the name assigned to that drive directory. The second column, Passwds Reqd, shows that password protection has been turned on for that drive.

As described in the SET command, each file can have up to two time stamps. The first of these time stamps can be either the creation date and time for the file or the date and time of the last access to the file. Access is defined as reading from or writing to the file. The third column of the SHOW [LABEL] output displays both the type of stamp and whether or not it is on. In the preceding example, *creation time* stamps are given to new files as shown by the stamp create column heading.

The fourth column displays the status of the second time stamp field, the update time stamp. Update time stamps display the date and time of the last update to a file, that is, the last time someone wrote to the file. In the SHOW [LABEL] display, update time stamps are turned on.

Besides showing the password protection and the active time stamps on a drive, SHOW [LABEL] also displays the date and time that the label was created and last updated.

## Display User Number Information

**Syntax:** SHOW {d:}[USERS]

**Explanation:** The preceding command displays the current user number and all the users on the drive and the corresponding number of files assigned to them.

**Example:**

```
A>SHOW [USERS]
Active User :     1
Active Files:     0    2    3    4
A: # of files:   95   40    1   26
A: Number of free directory entries: 350
A>
```

## Display Number of Free Directory Entries

**Syntax:** SHOW {d:}[DIR]

**Explanation:** The preceding command displays the number of free directory entries on the specified drive.

**Example:**

```
A>SHOW C:[DIR]

C: Number of free directory entries: 24

A>
```

The preceding command shows that there are only 24 free directory entries on drive C.

## Display Drive Characteristics

**Syntax:** SHOW {d:}[DRIVE]

**Explanation:** The preceding form of the SHOW command displays the drive characteristics of the specified drive.

**Example:** `A>SHOW [DRIVE]`

The following is an example of the system display for the preceding command:

```
    A: Drive Characteristics
3,600: 128 Byte Record Capacity
  450: Kilobyte Drive Capacity
   96: 32 Byte Directory Entries
   96: Checked Directory Entries
  128: Records / Directory Entry
   16: Records / Block
   48: Sectors / Track
  512: Bytes / Physical Record
```

# The SID Command

**Syntax:** SID {pgm-filespec} {,sym-filespec}

**Explanation:** The SID (Symbolic Instruction Debugger) allows you to monitor and test programs developed for the 8080 microprocessor. SID supports real-time breakpoints, fully monitored execution, symbolic disassembly, assembly, and memory display and fill functions. Utility programs are supplied with CP/M 3 that can be dynamically loaded with SID to provide traceback and histogram facilities.

SID commands display memory and CPU registers and direct the breakpoint operations during the debugging session.

Without a file specification SID loads into memory without a test program. Use this form to examine memory or to write and test simple programs using the A command. You must not use the SID commands G, T, or U, described later, until you have first loaded a test program.

A SID command line with a pgm-filespec loads both SID and the test program into memory. If the filetype is omitted from the filespec, COM is assumed. SID optionally loads in a symbol table file specified by sym-filespec. The sym-filespec needs no filetype because SID looks for a file with filetype SYM. Use the C, G, T, or U command to begin execution of the test program under supervision of SID.

Use CTRL-S to halt the screen display. CTRL-Q restarts the display. Abort lengthy displays by typing any keyboard character. Use CTRL-C to exit from SID.

SID can address absolute memory locations through symbolic expressions. A symbolic expression evaluates to either an address or a data item.

A symbolic expression can be a name from a SYM file produced from your program by a CP/M Macro Assembler. When you precede the symbolic expression with a period, SID returns its address in hexadecimal. When you precede the symbolic expression with the at sign, @, SID returns the 16-bit value stored at that location and the next contiguous location. When you precede the symbolic expression with an equal sign, SID returns the 8-bit value stored at that location. For two-byte expressions, this is the low byte because the 8080 microprocessor stores the low value of a two-byte word first.

A symbolic expression can be a literal value in hex, decimal, or ASCII, as indicated in the following list:

- SID uses literal hex values as given, but truncates any digits in excess of four on the left. The leftmost digit is the most significant digit. The rightmost digit is the least significant digit.
- To indicate decimal values precede them with a pound sign, #. Decimal values that evaluate to more than four hex digits are evaluated as the modulo of hex value FFFF. For example, #65534 = FFFEH, while #65536 = 0001H.
- SID translates literal ASCII character strings between apostrophes to the hex value of the two rightmost ASCII characters.

You can combine symbolic expressions with the symbolic operators, + or −, to produce another symbolic expression. Symbolic expressions combined in this way can be used to calculate the offset of an indirectly addressed data item, for example a subscripted variable. A special up-arrow operator, ^, can reference the top-of-stack item. A string of n ^ operators can reference the nth stack item without changing stack content or the stack pointer.

Table 5-18 lists the SID commands with their corresponding parameters and options. The actual command letter is printed in **boldface.** The parameters are in lower-case and follow the command letter. Optional items are in braces. Replace the arguments with the appropriate symbolic expressions as listed. Where two symbolic expressions are needed, SID can calculate the second one from the first using the symbolic operators described previously.

Table 5-18. SID Commands

| *Name* | *Syntax* | *Meaning* |
|---|---|---|
| Assemble | **As** | Enter assembly language statements. s is the start address. |
| Call | Cs {b{,d}} | Call to memory location from SID. s is the called address, b is the value of the BC register pair, and d is the value of the DE register pair. |
| Display | D{W}{s}{,f} | Display memory in hex and ASCII. W specifies a 16-bit word format, s is the start address, and f is the finish address. |
| Load | Epgm-filespec {,sym-filespec} | Load program and symbol table for execution. |
| Load | E* sym-filespec | Load a symbol table file. |
| Fill | Fs,f,d | Fill memory with constant value. s is the start address, f is the finish address, and d is an 8-bit data item. |
| Go | G{p}{,a{,b}} | Begin execution. p is a start address, a is a temporary breakpoint, and b is a second temporary breakpoint. G0 exits SID by performing a warm boot. |
| Hex | H<br>Ha<br>Ha,b | Displays all symbols with addresses in hex. The first syntax displays hex, decimal, and ASCII values of a. The second syntax performs number and character conversion, where a is a symbolic expression, and the third syntax computes hex sum and difference of a and b, where a and b are symbolic expressions. |
| Input | Icommand tail | Input CCP command line. |
| List | L {s}{,f} | List 8080 mnemonic instructions. s is the start address, and f is the finish address. |

Table 5-18. (continued)

| *Name* | *Syntax* | *Meaning* |
|---|---|---|
| Move | Ms,h,d | Move memory block. s is the start address, h is the high address of the block, and d is the destination start address. |
| Pass | P{p{,c}} | Pass point set, reset, and display. p is a permanent breakpoint address, and c is initial value of pass counter. |
| Read | Rfilespec{,d} | Read code/symbols. d is an offset to each address. |
| Set | S{W}s | Set memory values. s is an address where value is sent, W is a 16-bit word. |
| Trace | T{n{,c}} | Trace program execution. n is the number of program steps, and c is the utility entry address. |
| Trace | T{W}{n{,c}} | Trace without call. W instructs SID not to trace subroutines, n is the number of program steps, and c is the utility entry address. |
| Untrace | U{W}{n{,c}} | Monitor execution without trace. n is the number of program steps, c is the utility entry address, W instructs SID not to trace subroutines. |
| Value | V | Display the value of the next available location in memory (NEXT), the next location after the largest file read in (MSZE), the current value of the program counter (PC), and the address of the end of available memory (END). |
| Write | Wfilespec{,s,f} | Write the contents of a contiguous block of memory to filespec. s is the start address, f is the finish address. |
| Examine | X{f}{r} | Examine/alter CPU state. f is flag bit C, E, I, M, or Z; r is register A, B, D, H, P or S. |

**Examples:** A>SID

In the preceding example CP/M 3 loads SID from drive A into memory. SID displays the # prompt when it is ready to accept commands.

A>B:SID SAMPLE.HEX

In the preceding example, CP/M 3 loads SID and the program file SAMPLE.HEX into memory from drive B. SID displays:

| NEXT | MSZE | PC | END |
|---|---|---|---|
| nnnn | mmmm | pppp | eeee |

In the preceding example, nnnn is a hexadecimal address of the next free location following the loaded program, and mmmm is the next location after the largest program. This is initially the same value as NEXT. pppp is the initial hexadecimal value of the the program counter. eeee is the hexadecimal address of the logical end of the TPA.

#DFE00+#128+5

In the preceding example the first pound sign, #, is the SID prompt. This SID command, D, displays the values stored in memory starting at address FE80 (FE00 + #128) and ending at address FE85 (FE80 + 5).

## SID Utilities

The SID utilities HIST.UTL and TRACE.UTL are special programs that operate with SID to provide additional debugging facilities. The mechanisms for system initialization, data collection, and data display are described in the *CP/M SID™ Symbolic Instruction Debugger User's Guide*. The following discussion illustrates how a utility is activated. You load the utility by naming it as a parameter when invoking SID:

SID filename.UTL

In the preceding example filename is the name of the utility. Following the initial sign-on, the utility can prompt you for additional debugging parameters.

The HIST utility creates a histogram (bar graph) showing the relative frequency of execution of code within selected program segments of the test program. The HIST utility allows you to monitor those sections of code that execute most frequently.

Upon start-up HIST prompts

```
TYPE HISTOGRAM BOUNDS
```

Enter the bounds in the following format:

aaaa,bbbb

for a histogram between locations aaaa and bbbb inclusive. Collect data in U or T mode, then display results.

The TRACE utility obtains a traceback of the instructions that led to a particular breakpoint address in a program under test. You can collect the addresses of up to 256 instructions between pass points in U or T modes.

# The SUBMIT Command

**Syntax:** SUBMIT {filespec} {argument} ... {argument}

**Explanation:** The SUBMIT command lets you execute a group or batch of commands from a SUB file, which is a file with filetype of SUB.

Usually, you enter commands one line at a time. If you must enter the same sequence of commands several times, you might find it easier to batch the commands together using the SUBMIT command. To do this, create a file and enter your commands in this file. The file is identified by the filename, and must have a filetype of SUB. When you issue the SUBMIT command, SUBMIT reads the file named by the filespec and prepares it for interpretation by CP/M 3. When the preparation is complete, SUBMIT sends the file to CP/M 3 line-by-line, as if you were typing each command.

The SUBMIT command executes the commands from a SUB file as if you are entering the commands from the keyboard.

You create the SUB file with the ED utility. It can contain CP/M 3 commands, nested SUBMIT commands, and input data for a CP/M 3 command or a program.

You can pass arguments to SUB files when you execute them. Each argument you enter is assigned to a parameter in the SUB file. The first argument replaces every occurrence of $1 in the file, the second argument replaces parameter $2, etc., up to parameter $9. For example, if your file START.SUB contains the following commands:

```
ERA $1.BAK
DIR $1
PIP $1=A:$2.COM
```

and you enter the following SUBMIT command:

```
A>SUBMIT START SAM TEX
```

the argument SAM is substituted for every $1 in the START.SUB file, and TEX for every occurrence of $2 in the START.SUB file. SUBMIT then creates a file with the parameter substitutions and executes this file. This file now contains the following commands:

```
ERA SAM.BAK
DIR SAM
PIP SAM=A:TEX.COM
```

If you enter fewer arguments in the SUBMIT command than parameters in the SUB file, the remaining parameters are not included in the commands.

If you enter more arguments in the SUBMIT command than parameters in the SUB file, the remaining arguments are ignored.

To include an actual dollar sign, $ in your SUB file, type two dollar signs, $$. SUBMIT replaces them with a single dollar sign when it substitutes an argument for a parameter in the SUB file. For example, if file AA.SUB contains line:

```
MAC $1 $$$2
```

and you enter the following SUBMIT command:

```
A>SUBMIT AA ZZ SZ
```

then the translated file contains the following:

```
MAC ZZ $SZ
```

## Program Input Lines in a SUB File

A SUB file can contain program input lines. Any program input is preceded by a less than sign, <, as in the following example:

```
PIP
<B:=*.ASM
<CON:=DUMP.ASM
<
DIR
```

The three lines after PIP are input lines to the PIP command. The third line consists only of the < sign, indicating a carriage return. The carriage return causes PIP to return to the system to execute the final DIR command.

If the program terminates before using all of the input, SUBMIT ignores the excess input lines and displays the following warning message:

```
Warning: Program input ignored
```

If the program requires more input than is in the SUB file, it expects you to enter the remaining input from the keyboard.

You can enter control characters in a SUB file by using the usual convention of preceding the control character by an up-arrow character, ↑, followed by the letter to be converted to a control character. To enter an actual ↑ character, use the combination ↑↑. This combination translates to a single ↑ in the same manner that $$ translates to a single $.

## The SUB File

The SUB file can contain the following types of lines:

- Any valid CP/M 3 command
- Any valid CP/M 3 command with SUBMIT parameters
- Any data input line
- Any program input line with parameters ($0 to $9)

CP/M 3 command lines cannot exceed 128 characters.

**Example:** The following lines illustrate the variety of lines that can be entered in a SUB file:

```
DIR
DIR *.BAK
MAC $1 $$$4
PIP LST:=$1.PRN[T$2 $3 $5]
DIR *.ASM
PIP
<B:=*.ASM
<CON:=DUMP.ASM
<
DIR B:
```

## Executing the SUBMIT Command

**Syntax:** SUBMIT
SUBMIT filespec
SUBMIT filespec argument ... argument

If you enter only SUBMIT, the system prompts for the rest of the command. You enter the filespec and arguments.

**Example:** `A>`*`SUBMIT`*

The system displays the following prompt. Enter filespec and arguments here, such as:

`Enter File to Submit:` *`START B TEX`*

Another example could be

`A>`*`SUBMIT SUBA`*

Still another example using parameters is

```
A>SUBMIT AA ZZ SZ
```

where AA is the SUB file AA.SUB, ZZ is the argument to replace any occurrences of $1 in the AA.SUB file and SZ is the argument to replace all occurrences of $2 in the AA.SUB file.

## The PROFILE.SUB Start-up File

Every time you turn on or reset your computer, CP/M 3 automatically looks for a special SUB file named PROFILE.SUB to execute. If it does not exist, then CP/M 3 resumes normal operation. If the PROFILE.SUB file exists, the system executes the commands in the file. This file is convenient to use if you regularly execute a set of commands before you do your regular session on the computer. For example, if you want to be sure that you always enter the current date and time on your computer before you enter any other commands, you can create the PROFILE.SUB file, with ED, and enter the DATE command as follows:

DATE SET

Then, whenever you bring up the system, the system executes the DATE command and prompts you to enter the date and time. By using this facility, you can be sure to execute a regular sequence of commands before starting your usual session.

# The TYPE Command

**Syntax:** TYPE {filespec {[PAGE]|[NO PAGE]}}

**Explanation:** The TYPE command displays the contents of an ASCII character file on your screen. The PAGE option displays the console listing in paged mode, which means that the console listing stops automatically after listing n lines of text, where n is usually the system default of 24 lines per page. (See the DEVICE command to set n to a different value.) Press any character to continue listing another n lines of text. Press CTRL-C to exit back to the system. PAGE is the default mode.

The NO PAGE option displays the console listing continuously.

If you do not enter a file specification in the TYPE command the system prompts for a filename with the message:

```
Enter filename:
```

Respond with the filespec of the file you want listed.

Tab characters occurring in the file named by the file specification are expanded to every eighth column position of your screen.

At any time during the display, you can interrupt the listing by pressing CTRL-S. Press CTRL-Q to resume the listing.

Press CTRL-C to exit back to the system.

Make sure the file specification identifies a file containing character data.

If the file named by the file specification is not present on the specified drive, TYPE displays the following message on your screen:

```
No File
```

To list the file at the printer and on the screen, type a CTRL-P before entering the TYPE command line. To stop echoing console output at the printer, type a second CTRL-P. The type command displays the contents of the file until the screen is filled. It then pauses until you press any key to continue the display.

**Examples:**

```
A>TYPE MYPROG.PLI
```

This command displays the contents of the file MYPROG.PLI on your screen, a page at a time.

```
A>TYPE B:THISFILE [NO PAGE]
```

This command continuously displays the contents of the file THISFILE from drive B on your screen.

# The USER Command

**Syntax:** USER {number}

**Explanation:** The USER command sets the current user number. When you start CP/M 3, 0 is the current user number. You can use a USER command to change the current user number to another in the range 0-15.

CP/M 3 identifies every file with a user number. In general, you can access only files identified with the current user number. However, if you mark a file in user 0 with the SYS attribute, the file can be accessed from all other user numbers.

**Examples:**

```
A>USER
```

The system command prompts for the user number, as follows:

```
Enter User#:5
5A>
```

The current user number is now 5 on drive A.

```
A>USER 3
3A>
```

This command changes the current user number to 3.

# The XREF Command

**Syntax:** XREF {d:} filename {$P}

**Explanation:** The XREF command provides a cross-reference summary of variable usage in a program. XREF requires the PRN and SYM files produced by MAC or RMAC for the program.

The SYM and PRN files must have the same filename as the filename in the XREF command tail. XREF outputs a file of type XRF.

**Examples:**

```
A>XREF b:MYPROG
```

In this example, XREF is on drive A. XREF operates on the file MYPROG.SYM and MYPROG.PRN which are on drive B. XREF produces the file MYPROG.XRF on drive B.

```
A>XREF b:MYPROG $P
```

In the preceding example, the $P option directs output to the printer.

*End of Section 5*

# Section 6
# ED, The CP/M 3 Context Editor

## 6.1 Introduction to ED

To do almost anything with a computer you need some way to enter data, a way to give the computer the information you want it to process. The programs most commonly used for this task are called editors. They transfer your keystrokes at the keyboard to a disk file. CP/M 3's editor is named ED. Using ED, you can easily create and alter CP/M 3 text files.

The correct command format for invoking the CP/M 3 editor is given in Section 6.2, "Starting ED." After starting ED, you issue commands that transfer text from a disk file to memory for editing. Section 6.3, "ED Operation," details this operation and describes the basic text transfer commands that allow you to easily enter and exit the editor.

Section 6.4, "Basic Editing Commands," details the commands that edit a file. Section 6.5, "Combining ED Commands," describes how to combine the basic commands to edit more efficiently. Although you can edit any file with the basic ED commands, ED provides several more commands that perform more complicated editing functions, as described in Section 6.6, "Advanced ED Commands."

During an editing session, ED can return two types of error messages. Section 6.7, "ED Error Messages," lists these messages and provides examples that indicate how to recover from common editing error conditions.

## 6.2 Starting ED

Syntax:

    ED input-filespec {d: | output-filespec}

To start ED, enter its name after the CP/M 3 prompt. The command ED must be followed by a file specification, one that contains no wildcard characters, such as:

```
A>ED MYFILE.TEX
```

The file specification, MYFILE.TEX in the preceding example, specifies a file to be edited or created. The file specification can be preceded by a drive specification, but a drive specification is unnecessary if the file to be edited is on your default drive. Optionally, the file specification can be followed by a drive specification, as shown in the following example:

```
A>ED MYFILE.TEX B:
```

In response to this command, ED opens the file to be edited, MYFILE.TEX, on drive A, but sends all the edited material to a file on drive B.

Optionally, you can send the edited material to a file with a different filename, as in the following example:

```
A>ED MYFILE.TEX YOURFILE.TEX
```

The file with the different filename cannot already exist or ED prints the following message and terminates.

```
Output File Exists, Erase It
```

The ED prompt, *, appears at the screen when ED is ready to accept a command, as follows

```
A>ED MYFILE.TEX
      : *
```

If no previous version of the file exists on the current disk, ED automatically creates a new file and displays the following message:

```
NEW FILE
      : *
```

**Note:** before starting an editing session, use the SHOW command to check the amount of free space on your disk. Make sure that the unused portion of your disk is at least as large as the file you are editing, or larger if you plan to add characters to the file. When ED finds a disk or directory full, ED has only limited recovery mechanisms. These are explained in Section 6.7, "ED Error Messages."

## 6.3 ED Operation

With ED, you change portions of a file that pass through a memory buffer. When you start ED with one of the preceding commands, this memory buffer is empty. At your command, ED reads segments of the source file, for example MYFILE.TEX, into the memory buffer for you to edit. If the file is new, you must insert text into the file before you can edit. During the edit, ED writes the edited text onto a temporary work file, MYFILE.$$$.

When you end the edit, ED writes the memory buffer contents to the temporary file, followed by any remaining text in the source file. ED then changes the name of the source file from MYFILE.TEX to MYFILE.BAK, so you can reclaim this original material from the back-up file if necessary. ED then renames the temporary file, MYFILE.$$$, to MYFILE.TEX, the new edited file. The following figure illustrates the relationship between the source file, the temporary work file, and the new file.

**Note:** when you invoke ED with two filespecs, an input file and an output file, ED does not rename the input file to type BAK; therefore, the input file can be Read-Only or on a write-protected disk if the output file is written to another disk.

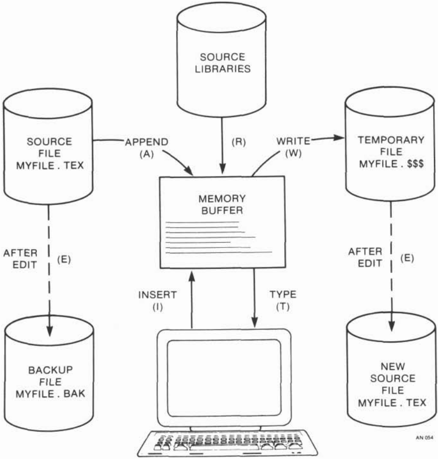


**Figure 6-1. Overall ED Operation**

In the preceding figure, the memory buffer is logically between the source file and the temporary work file. ED supports several commands that transfer lines of text between the source file, the memory buffer, and the temporary, and eventually final, file. The following table lists the three basic text transfer commands that allow ~~you~~ to easily enter the editor, write text to the temporary file, and exit the editor.

Table 6-1. Text Transfer Commands

| *Command* | *Result* |
|---|---|
| nA | Append the next n unprocessed source lines from the source file to the end of the memory buffer. |
| nW | Write the first n lines of the memory buffer to the temporary file free space. |
| E | End the edit. Copy all buffered text to the temporary file, and copy all unprocessed source lines to the temporary file. Rename files. |

### 6.3.1 Appending Text into the Buffer

When you start ED and the memory buffer is empty, you can use the A (append) command to add text to the memory buffer.

**Note:** ED can number lines of text to help you keep track of data in the memory buffer. The colon that appears when you start ED indicates that line numbering is turned on. Type -V after the ED prompt to turn the line number display off. Line numbers appear on the screen but never become a part of the output file.

The V (Verify Line Numbers) Command

The V command turns the line number display in front of each line of text on or off. The V command also displays the free bytes and total size of the memory buffer. The V command takes the following forms:

V, -V, 0V

Initially, the line number display is on. Use -V to turn it off. If the memory buffer is empty, or if the current line is at the end of the memory buffer, ED represents the line number as five blanks. The 0V command prints the memory buffer statistics in the form:

free/total

where free is the number of free bytes in the memory buffer, and total is the size of the memory buffer. For example, if you have a total of 48,253 bytes in the memory buffer and 46,652 of them are free, the 0V command displays this information as follows

```
46652/48253
```

If the buffer is full, the first field, which indicates free space, is blank.

The A (Append) Command

The A command appends, copies, lines from an existing source file into the memory buffer. The A command takes the following form:

nA

where n is the number of unprocessed source lines to append into the memory buffer. If a pound sign, #, is given in place of n, then the integer 65,535 is assumed. Because the memory buffer can contain most reasonably sized source files, it is often possible to issue the command #A at the beginning of the edit to read the entire source file into memory.

When n is 0, ED appends the unprocessed source lines into the memory buffer until the buffer is approximately half full. If you do not specify n, ED appends one line from the source file into the memory buffer.

### 6.3.2 ED Exit

You can use the W (Write) command and the E (Exit) command to save your editing changes. The W command writes lines from the memory buffer to the new file without ending the ED session. An E command saves the contents of the buffer and any unprocessed material from the source file and exits ED.

## The W (Write) Command

The W command writes lines from the buffer to the new file. The W command takes the form:

nW

where n is the number of lines to be written from the beginning of the buffer to the end of the new file. If n is greater than 0, ED writes n lines from the beginning of the buffer to the end of the new file. If n is 0, ED writes lines until the buffer is half empty. The 0W command is a convenient way of making room in the memory buffer for more lines from the source file. If the buffer is full, you can use the 0W command to write half the contents of the memory buffer to the new file. You can use the #W command to write the entire contents of the buffer to the new file. Then you can use the 0A command to read in more lines from the source file.

**Note:** after a W command is executed, you must enter the H command to reedit the saved lines during the current editing session.

## The E (Exit) Command

An E command performs a normal exit from ED. The E command takes the form:

E

followed by a carriage return.

When you enter an E command, ED first writes all data lines from the buffer and the original source file to the $$$ file. If a BAK file exists, ED deletes it, then renames the original file with the BAK filetype. Finally, ED renames the $$$ file from filename.$$$ to the original filetype and returns control to the operating system.

The operation of the E command makes it unwise to edit a back-up file. When you edit a BAK file and exit with an E command, ED erases your original file because it has a BAK filetype. To avoid this, always rename a back-up file to some other filetype before editing it with ED.

**Note:** any command that terminates an ED session must be the only command on the line.

## 6.4 Basic Editing Commands

The text transfer commands discussed previously allow you to easily enter and exit the editor. This section discusses the basic commands that edit a file.

ED treats a file as a long chain of characters grouped together in lines. ED displays and edits characters and lines in relation to an imaginary device called the character pointer (CP). During an edit session, you must mentally picture the CP's location in the memory buffer and issue commands to move the CP and edit the file.

The following commands move the character pointer or display text in the vicinity of the CP. These ED commands consist of a numeric argument and a single command letter and must be followed by a carriage return. The numeric argument, n, determines the number of times ED executes a command; however, there are four special cases to consider in regard to the numeric argument:

- If the numeric argument is omitted, ED assumes an argument of 1.
- Use a negative number if the command is to be executed backwards through the memory buffer. The B command is an exception.
- If you enter a pound sign, #, in place of a number, ED uses the value 65,535 as the argument. A pound sign argument can be preceded by a minus sign to cause the command to execute backwards through the memory buffer, -#.
- ED accepts 0 as a numeric argument only in certain commands. In some cases, 0 causes the command to be executed approximately half the possible number of times, while in other cases it prevents the movement of the CP.

The following table alphabetically summarizes the basic editing commands and their valid arguments.

Table 6-2. Basic Editing Commands

| *Command* | *Action* |
|---|---|
| B, -B | Move CP to the beginning (B) or end (-B) of the memory buffer. |
| nC, -nC | Move CP n characters forward (nC) or backward (-nC) through the memory buffer. |
| nD, -nD | Delete n characters before (-nD) or after (nD) the CP. |
| I | Enter insert mode. |
| Istring CTRL-Z | Insert a string of characters. |
| nK, -nK | Delete (kill) n lines before the CP (-nK) or after the CP (nK). |
| nL, -nL | Move the CP n lines forward (nL) or backward (-nL) through the memory buffer. |
| nT, -nT | Type n lines before the CP (-nT) or after the CP (nT). |
| n, -n | Move the CP n lines before the CP (-n) or after the CP (n) and display the destination line. |

The following sections discuss ED's basic editing commands in more detail. The examples in these sections illustrate how the commands affect the position of the character pointer in the memory buffer. Later examples in Section 6.5, "Combining ED Commands," illustrate how the commands appear at the screen. For these sections, however, the symbol ^ in command examples represents the character pointer, which you must imagine in the memory buffer.

### 6.4.1 Moving the Character Pointer

This section describes commands that move the character pointer in useful increments but do not display the destination line. Although ED is used primarily to create and edit program source files, the following sections present a simple text as an example to make ED easier to learn and understand.

#### The B (Beginning/Bottom) Command

The B command moves the CP to the beginning or bottom of the memory buffer. The B command takes the following forms:

B, -B

-B moves the CP to the end or bottom of the memory buffer; B moves the CP to the beginning of the buffer.

#### The C (Character) Command

The C command moves the CP forward or backward the specified number of characters. The C command takes the following forms:

nC, -nC

when n is the number of characters the CP is to be moved. A positive number moves the CP towards the end of the line and the bottom of the buffer. A negative number moves the CP towards the beginning of the line and the top of the buffer. You can enter an n large enough to move the CP to a different line. However, each line is separated from the next by two invisible characters: a carriage return and a line-feed, represented by <cr><lf>. You must compensate for their presence. For example, if the CP is pointing to the beginning of the line, the command 30C moves the CP to the next line:

Emily Dickinson said,<cr><lf>
"I fin^d ecstasy in living —<cr><lf>

## The L (Line) Command

The L command moves the CP the specified number of lines. After an L command, the CP always points to the beginning of a line. The L command takes the following forms:

nL, -nL

where n is the number of lines the CP is to be moved. A positive number moves the CP towards the end of the buffer. A negative number moves the CP back toward the beginning of the buffer. The command 2L moves the CP two lines forward through the memory buffer and positions the character pointer at the beginning of the line.

"I find ecstasy in living —<cr><lf>
the mere sense of living<cr><lf>
^is joy enough."<cr><lf>

The command -L moves the CP to the beginning of the previous line, even if the CP originally points to a character in the middle of the line. Use the special character 0 to move the CP to the beginning of the current line.

## The n (Number) Command

The n command moves the CP and displays the destination line. The n command takes the following forms:

n, -n

where n is the number of lines the CP is to be moved. In response to this command, ED moves the CP forward or backward the number of lines specified, then prints only the destination line. For example, the command -2 moves the CP back two lines.

Emily Dickinson said,<cr><lf>
^"I find ecstasy in living —<cr><lf>
the mere sense of living<cr><lf>
is joy enough."<cr><lf>

A further abbreviation of this command is to enter no number at all. In response to a carriage return without a preceding command, ED assumes a n command of 1 and moves the CP down to the next line and prints it, as follows

Emily Dickinson said,<cr><lf>
"I find ecstasy in living —<cr><lf>
^the mere sense of living<cr><lf>

Also, a minus sign, −, without a number moves the CP back one line.

### 6.4.2 Displaying Memory Buffer Contents

ED does not display the contents of the memory buffer until you specify which part of the text you want to see. The T command displays text without moving the CP.

The T (Type) Command

The T command types a specified number of lines from the CP at the screen. The T command takes the forms

nT, -nT

where n specifies the number of lines to be displayed. If a negative number is entered, ED displays n lines before the CP. A positive number displays n lines after the CP. If no number is specified, ED types from the character pointer to the end of the line. The CP remains in its original position no matter how many lines are typed. For example, if the character pointer is at the beginning of the memory buffer, and you instruct ED to type four lines (4T), four lines are displayed at the screen, but the CP stays at the beginning of line 1.

^Emily Dickinson said,<cr><lf>
"I find ecstasy in living —<cr><lf>
the mere sense of living<cr><lf>
is joy enough."<cr><lf>

If the CP is between two characters in the middle of the line, a T command with no number specified types only the characters between the CP and the end of the line, but the character pointer stays in the same position, as shown in the following memory buffer example:

```
"I find ec^stasy in living -
```

Whenever ED is displaying text with the T command, you can enter a CTRL-S to stop the display, then press any key when you are ready to continue scrolling. Enter a CTRL-C to abort long type-outs.

### 6.4.3 Deleting Characters

The D (Delete) Command

The D command deletes a specified number of characters and takes the forms:

```
nD, -nD
```

where n is the number of characters to be deleted. If no number is specified, ED deletes the character to the right of the CP. A positive number deletes multiple characters to the right of the CP, towards the bottom of the file. A negative number deletes characters to the left of the CP, towards the top of the file. If the character pointer is positioned in the memory buffer as follows

```
Emily Dickinson said,<cr><lf>
"I find ecstasy in living —<cr><lf>
the mere sense of living<cr><lf>
is joy ^enough."<cr><lf>
```

the command 6D deletes the six characters after the CP, and the resulting memory buffer looks like this:

```
Emily Dickinson said,<cr><lf>
"I find ecstasy in living —<cr><lf>
the mere sense of living<cr><lf>
is joy^."<cr><lf>
```

You can also use a D command to delete the <cr><lf> between two lines to join them together. Remember that the <cr> and <lf> are two characters.

The K (Kill) Command

The K command kills or deletes whole lines from the memory buffer and takes the forms:

nK, -nK

where n is the number of lines to be deleted. A positive number kills lines after the CP. A negative number kills lines before the CP. When no number is specified, ED kills the current line. If the character pointer is at the beginning of the second line,

Emily Dickinson said,<cr><lf>
"I find ecstasy in living —<cr><lf>
the mere sense of living<cr><lf>
is joy enough."<cr><lf>

then the command -K deletes the previous line and the memory buffer changes:

"I find ecstasy in living —<cr><lf>
the mere sense of living<cr><lf>
is joy enough."<cr><lf>

If the CP is in the middle of a line, a K command kills only the characters from the CP to the end of the line and concatenates the characters before the CP with the next line. A -K command deletes all the characters between the beginning of the previous line and the CP. A 0K command deletes the characters on the line up to the CP.

You can use the special # character to delete all the text from the CP to the beginning or end of the buffer. Be careful when using #K because you cannot reclaim lines after they are removed from the memory buffer.

### 6.4.4 Inserting Characters into the Memory Buffer

The I (Insert) Command

To insert characters into the memory buffer from the screen, use the I command.

If you enter the command in upper-case, ED automatically converts the string to upper-case. The I command takes the forms:

I
Istring^Z

When you type the first command, ED enters insert mode. In this mode, all keystrokes are added directly to the memory buffer. ED enters characters in lines and does not start a new line until you press the enter key.

```
A>ED B:QUOTE.TEX

  NEW FILE
     : *i
    1:  Emily Dickinson said,
    2:  "I find ecstasy in living -
    3:  the mere sense of living
    4:  is joy enough."
    5:  ^Z
     : *
```

**Note:** to exit from insert mode, you must press CTRL-Z or ESC. When the ED prompt, *, appears on the screen, ED is not in insert mode.

In command mode, you can use CP/M 3 command line-editing control characters. In insert mode, you can use the control characters listed in Table 6-3.

**Table 6-3. CP/M 3 Line-editing Controls**

| *Command* | *Result* |
|---|---|
| CTRL-H | Delete the last character typed on the current line. |
| CTRL-U | Delete the entire line currently being typed. |
| CTRL-X | Delete the entire line currently being typed. Same as CTRL-U. |
| Backspace | Remove the last character. |

When entering a combination of numbers and letters, you might find it inconvenient to press a caps-lock key if your terminal translates the upper-case of numbers to special characters. ED provides two ways to translate your alphabetic input to upper-case without affecting numbers. The first is to enter the insert command letter in upper-case: I. All alphabetics entered during the course of the capitalized command, either in insert mode or as a string, are translated to upper-case. If you enter the insert command letter in lower-case, all alphabetics are inserted as typed. The second method is to enter a U command before inserting text. Upper-case translation remains in effect until you enter a -U command.

The Istring^Z (Insert String) Command

The second form of the I command does not enter insert mode. It inserts the character string into the memory buffer and returns immediately to the ED prompt. You can use CP/M 3's line-editing control characters to edit the command string.

To insert a string, first use one of the commands that position the CP. You must move the CP to the place where you want to insert a string. For example, if you want to insert a string at the beginning of the first line, use a B command to move the CP to the beginning of the buffer. With the CP positioned correctly, enter an insert string, as follows

```
iIn 1870, ^Z
```

This inserts the phrase "In 1870," at the beginning of the first line, and returns immediately to the ED prompt. In the memory buffer, the CP appears after the inserted string, as follows

In 1870,^ Emily Dickinson said,<cr><lf>

### 6.4.5 Replacing Characters

The S (Substitute) Command

The S command searches the memory buffer for the specified string, but when it finds it, automatically substitutes a new string for the search string. Whenever you enter a command in upper-case, ED automatically converts the string to upper-case. The S command takes the form:

nSsearch string^Znew string

where n is the number of substitutions to make. If no number is specified, ED searches for the next occurrence of the search string in the memory buffer. For example, the command

```
sEmily Dickinson^ZThe poet
```

searches for the first occurrence of "Emily Dickinson" and substitutes "The poet." In the memory buffer, the CP appears after the substituted phrase, as follows

The poet^ said,<cr><lf>

If upper-case translation is enabled by a capital S command letter, ED looks for a capitalized search string and inserts a capitalized insert string. Note that if you combine this command with other commands, you must terminate the new string with a CTRL-Z.

## 6.5 Combining ED Commands

It saves keystrokes and editing time to combine the editing and display commands. You can type any number of ED commands on the same line. ED executes the command string only after you press the carriage return key. Use CP/M 3's line-editing controls to manipulate ED command strings.

When you combine several commands on a line, ED executes them in the same order they are entered, from left to right on the command line. There are four restrictions to combining ED commands:

- The combined-command line must not exceed CP/M 3's 128 character maximum.
- If the combined-command line contains a character string, the line must not exceed 100 characters.
- Commands to terminate an editing session must not appear in a combined-command line.
- Commands, such as the I, J, R, S, and X commands, that require character strings or filespecs must be either the last command on a line or must be terminated with a CTRL-Z or ESC character, even if no character string or filespec is given.

While the examples in the previous section show the memory buffer and the position of the character pointer, the examples in this section show how the screen looks during an editing session. Remember that the character pointer is imaginary, but you must picture its location because ED's commands display and edit text in relation to the character pointer.

### 6.5.1 Moving the Character Pointer

To move the CP to the end of a line without calculating the number of characters, combine an L command with a C command, L-2C. This command string accounts for the <cr><lf> sequence at the end of the line.

Change the C command in this command string to move the CP more characters to the left. You can use this command string if you must make a change at the end of the line and you do not want to calculate the number of characters before the change, as in the following example:

```
    1: *T
    1:  Emily Dickinson said,
    1: *L-7CT
said,
    1: *
```

### 6.5.2 Displaying Text

A T command types from the CP to the end of the line. To see the entire line, you can combine an L command and a T command. Type 0lt to move the CP from the middle to the beginning of the line and then display the entire line. In the following example, the CP is in the middle of the line. 0L moves the CP to the beginning of the line. T types from the CP to the end of the line, allowing you to see the entire line.

```
    3: *T
sense of living
    3: *0LT
    3:  the mere sense of living
    3: *
```

The command 0TT displays the entire line without moving the CP.

To verify that an ED command moves the CP correctly, combine the command with the T command to display the line. The following example combines a C command and a T command.

```
    2: *8CT
ecstasy in living -
    2: *
```

```
    4: *B#T
    1:  Emily Dickinson said,
    2:  "I find ecstasy in living -
    3:  the mere sense of living
    4:  is joy enough."
    1: *
```

### 6.5.3 Editing

To edit text and verify corrections quickly, combine the edit commands with other ED commands that move the CP and display text. Command strings like the one that follows move the CP, delete specified characters, and verify changes quickly.

```
1: *15C5D0LT
1:  Emily Dickinson,
1: *
```

Combine the edit command K with other ED commands to delete entire lines and verify the correction quickly, as follows

```
1: *2L2KB#T
1:  Emily Dickinson said,
2:  "I find ecstasy in living -
1: *
```

The abbreviated form of the I (insert) command makes simple textual changes. To make and verify these changes, combine the I command string with the C command and the 0LT command string as follows. Remember that the insert string must be terminated by a CTRL-Z.

```
1: *20Ci to a friend^Z0LT
1:  Emily Dickinson said to a friend,
1: *
```

## 6.6 Advanced ED Commands

The basic editing commands discussed previously allow you to use ED for all your editing. The following ED commands, however, enhance ED's usefulness.

### 6.6.1 Moving the CP and Displaying Text

The P (Page) Command

Although you can display any amount of text at the screen with a T command, it is sometimes more convenient to page through the buffer, viewing whole screens of data and moving the CP to the top of each new screen at the same time. To do this, use ED's P command. The P command takes the following forms:

    nP, -nP

where n is the number of pages to be displayed. If you do not specify n, ED types the 23 lines following the CP and then moves the CP forward 23 lines. This leaves the CP pointing to the first character on the screen.

To display the current page without moving the CP, enter 0P. The special character 0 prevents the movement of the CP. If you specify a negative number for n, P pages backwards towards the top of the file.

The n: (Line Number) Command

When line numbers are being displayed, ED accepts a line number as a command to specify a destination for the CP. The line number command takes the following form:

    n:

where n is the number of the destination line. This command places the CP at the beginning of the specified line. For example, the command 4: moves the CP to the beginning of the fourth line.

Remember that ED dynamically renumbers text lines in the buffer each time a line is added or deleted. Therefore, the number of the destination line you have in mind can change during editing.

The :n (Through Line Number) Command

The inverse of the line number command specifies that a command should be executed through a certain line number. You can use this command with only three ED commands: the K (kill) command, the L (line) command, and the T (type) command. The :n command takes the following form:

:ncommand

where n is the line number through which the command is to be executed. The :n part of the command does not move the CP, but the command that follows it might.

You can combine n: with :n to specify a range of lines through which a command should be executed. For example, the command 2::4T types the second, third, and fourth lines:

```
1: *2::4T
2:  "I find ecstasy in living -
3:  the mere sense of living
4:  is joy enough."
2: *
```

### 6.6.2 Finding and Replacing Character Strings

ED supports a find command, F, that searches through the memory buffer and places the CP after the word or phrase you want. The N command allows ED to search through the entire source file instead of just the buffer. The J command searches for and then juxtaposes character strings.

The F (Find) Command

The F command performs the simplest find function; it takes the form:

nFstring

where n is the occurrence of the string to be found. Any number you enter must be positive because ED can only search from the CP to the bottom of the buffer. If you enter no number, ED finds the next occurrence of the string in the file. In the following example, the second occurrence of the word living is found.

```
1: *2fliving
3: *
```

The character pointer moves to the beginning of the third line where the second occurrence of the word "living" is located. To display the line, combine the find command with a type command. Note that if you follow an F command with another ED command on the same line, you must terminate the string with a CTRL-Z, as follows

```
1: *2fliving^Z0lt
3: *the mere sense of living
```

It makes a difference whether you enter the F command in upper- or lower-case. If you enter F, ED internally translates the argument string to upper-case. If you specify f, ED looks for an exact match. For example, Fcp/m 3 searches for CP/M 3 but fcp/m 3 searches for cp/m 3, and cannot find CP/M 3.

If ED does not find a match for the string in the memory buffer, it issues the message,

```
BREAK "#" AT
```

where the symbol # indicates that the search failed during the execution of an F command.

### The N Command

The N command extends the search function beyond the memory buffer to include the source file. If the search is successful, it leaves the CP pointing to the first character after the search string. The N command takes the form:

nNstring

where n is the occurrence of the string to be found. If no number is entered, ED looks for the next occurrence of the string in the file. The case of the N command has the same effect on an N command as it does on an F command. Note that if you follow an N command with another ED command, you must terminate the string with a CTRL-Z.

When an N command is executed, ED searches the memory buffer for the specified string, but if ED does not find the string, it does not issue an error message. Instead, ED automatically writes the searched data from the buffer into the new file. Then ED performs a 0A command to fill the buffer with unsearched data from the source file. ED continues to search the buffer, write out data, and append new data until it either finds the string or reaches the end of the source file. If ED reaches the end of the source file, ED issues the following message:

```
BREAK "#" AT
```

Because ED writes the searched data to the new file before looking for more data in the source file, ED usually writes the contents of the buffer to the new file before finding the end of the source file and issuing the error message.

**Note:** you must use the H command to continue an edit session after the source file is exhausted and the memory buffer is emptied.

### The J (Juxtapose) Command

The J command inserts a string after the search string, then deletes any characters between the end of the inserted string to the beginning of the a third delete-to string. This juxtaposes the string between the search and delete-to strings with the insert string. The J command takes the form:

nJsearch string^Zinsert string^Zdelete-to string

where n is the occurrence of the search string. If no number is specified, ED searches for the next occurrence of the search string in the memory buffer. In the following example, ED searches for the word "Dickinson", inserts the phrase "told a friend" after it, and then deletes everything up to the comma.

```
1: *#T
1:  Emily Dickinson said,
2:  "I find ecstasy in living -
3:  the mere of living
4:  is joy enough."
1: *JDickinson^Z told a friend^Z,
1: *0lt
1:  Emily Dickinson told a friend,
1: *
```

If you combine this command with other commands, you must terminate the delete-to string with a CTRL-Z or ESC, as in the following example. If an upper-case J command letter is specified, ED looks for upper-case search and delete-to strings and inserts an upper-case insert string.

The J command is especially useful when revising comments in assembly language source code, as follows

```
236:  SORT    LXI    H, SW    ;ADDRESS TOGGLE SWITCH
236: *J;^ZADDRESS SWITCH TOGGLE^Z^L^ZOLT
236:  SORT    LXI    H, SW    ;ADDRESS SWITCH TOGGLE
236: *
```

In this example, ED searches for the first semicolon and inserts ADDRESS SWITCH TOGGLE after the mark and then deletes to the <cr><lf> sequence, represented by CTRL-L. In any search string, you can use CTRL-L to represent a <cr><lf> when the phrase that you want extends across a line break. You can also use a CTRL-I in a search string to represent a tab.

**Note:** if long strings make your command longer than your screen line length, enter a CTRL-E to cause a physical carriage return at the screen. A CTRL-E returns the cursor to the left edge of the screen, but does not send the command line to ED. Remember that no ED command line containing strings can exceed 100 characters. When you finish your command, press the carriage return key to send the command to ED.

### The M (Macro) Command

An ED macro command, M, can increase the usefulness of a string of commands. The M command allows you to group ED commands together for repeated execution. The M command takes the following form:

nMcommand string

where n is the number of times the command string is to be executed. A negative number is not a valid argument for an M command. If no number is specified, the special character # is assumed, and ED executes the command string until it reaches

the end of data in the buffer or the end of the source file, depending on the commands specified in the string. In the following example, ED executes the four commands repetitively until it reaches the end of the memory buffer:

```
        1: *mfliving^Z-6diLiving^Z0lt
        2:   "I find ecstasy in Living -
        3:   the mere sense of Living

    BREAK "#" AT ^Z
3: *
```

The terminator for an M command is a carriage return; therefore, an M command must be the last command on the line. Also, all character strings that appear in a macro must be terminated by CTRL-Z or ESC. If a character string ends the combined-command string, it must be terminated by CTRL-Z, then followed by a <cr> to end the M command.

The execution of a macro command always ends in a `BREAK"#"` message, even when you have limited the number of times the macro is to be performed, and ED does not reach the end of the buffer or source file. Usually the command letter displayed in the message is one of the commands from the string and not M.

To abort a macro command, press a CTRL-C at the keyboard.

### The Z (Sleep) Command

Use the Z command to make the editor pause between operations. The pauses give you a chance to review what you have done. The Z command takes the following form:

nZ

where n is the number of seconds to wait before proceeding to the next instruction.

Usually, the Z command has no real effect unless you use it with a macro command. The following example shows you how you can use the Z command to cause a brief pause each time ED finds the word TEXT in a file.

```
A>*mfliving^Z0tt10z
```

### 6.6.3 Moving Text Blocks

To move a group of lines from one area of your data to another, use an X command to write the text block into a temporary LIB file, then a K command to remove these lines from their original location, and finally an R command to read the block into its new location.

The X (Transfer) Command

The X command takes the forms:

nX
nXfilespec^Z

where n is the number of lines from the CP towards the bottom of the buffer that are to be transferred to a file. Therefore, n must always be a positive number. The nX command with no file specified creates a temporary file named X$$$$$$$.LIB. This file is erased when you terminate the edit session. The nX command with a file specified creates a file of the specified name. If no filetype is specified, LIB is assumed. This file is saved when you terminate the edit session. If the X command is not the last command on the line, the command must be terminated by a CTRL-Z or ESC. In the following example, just one line is transferred to the temporary file:

```
1: *X
1: *t
1: *Emily Dickinson said,
1: *Kt
1: *"I find ecstasy in living -
1: *
```

If no library file is specified, ED looks for a file named X$$$$$$$.LIB. If the file does not exist, ED creates it. If a previous X command already created the library file, ED appends the specified lines to the end of the existing file.

Use the special character 0 as the n argument in an X command to delete any file from within ED.

The R (Read) Command

The X command transfers the next n lines from the current line to a library file. The R command can retrieve the transferred lines. The R command takes the forms:

R
Rfilepsec

If no filename is specified, X$$$$$$$ is assumed. If no filetype is specified, LIB is assumed. R inserts the library file in front of the CP; therefore, after the file is added to the memory buffer, the CP points to the same character it did before the read, although the character is on a new line number. If you combine an R command with other commands, you must separate the filename from subsequent command letters with a CTRL-Z as in the following example where ED types the entire file to verify the read.

```
1: *41
 : *R^ZB#T
1:   "I find ecstasy in living -
2:   the mere sense of living
3:   is joy enough."
4:   Emily Dickinson said,
1: *
```

### 6.6.4 Saving or Abandoning Changes: ED Exit

You can save or abandon editing changes with the following three commands.

The H (Head of File) Command

An H command saves the contents of the memory buffer without ending the ED session, but it returns to the head of the file. It saves the current changes and lets you reedit the file without exiting ED. The H command takes the following form:

```
H
```

followed by a carriage return.

To execute an H command, ED first finalizes the new file, transferring all lines remaining in the buffer and the source file to the new file. Then ED closes the new file, erases any BAK file that has the same file specification as the original source file, and renames the original source file filename.BAK. ED then renames the new file, which has had the filetype $$$, with the original file specification. Finally, ED opens the newly renamed file as the new source file for a new edit, and opens a new $$$ file. When ED returns the * prompt, the CP is at the beginning of an empty memory buffer.

If you want to send the edited material to a file other than the original file, use the following command:

```
A>ED filespec differentfilespec
```

If you then restart the edit with the H command, ED renames the file differentfilename.$$$ to differentfilename.BAK and creates a new file of differentfilespec when you finish editing.

The O (Original) Command

An O command abandons changes made since the beginning of the edit and allows you to return to the original source file and begin reediting without ending the ED session. The O command takes the form:

O

followed by a carriage return. When you enter an O command, ED confirms that you want to abandon your changes by asking

```
O (Y/N)?
```

You must respond with either a Y or an N; if you press any other key, ED repeats the question. When you enter Y, ED erases the temporary file and the contents of the memory buffer. When the * prompt returns, the character pointer is pointing to the beginning of an empty memory buffer, just as it is when you start ED.

### The Q (Quit) Command

A Q command abandons changes made since the beginning of the ED session and exits ED. The Q command takes the form:

Q

followed by a carriage return.

When you enter a Q command, ED verifies that you want to abandon the changes by asking

```
Q (Y/N)?
```

You must respond with either a Y or an N; if you press any other key, ED repeats the question. When you enter Y, ED erases the temporary file, closes the source file, and returns control to CP/M 3.

**Note:** you can enter a CTRL-Break or a CTRL-C to return control immediately to CP/M 3. This does not give ED a chance to close the source or new files, but it prevents ED from deleting any temporary files.

## 6.7 ED Error Messages

ED returns one of two types of error messages: an ED error message if ED cannot execute an edit command, or a CP/M 3 error message if ED cannot read or write to the specified file. An ED error message takes the form:

BREAK "x" AT c

where x is one of the symbols defined in the following table and c is the command letter where the error occurred.

Table 6-4. ED Error Symbols

| *Symbol* | *Meaning* |
|---|---|
| # | Search failure. ED cannot find the string specified in a F, S, or N command. |
| ?c | Unrecognized command letter c. ED does not recognize the indicated command letter, or an E, H, O, or Q command is not alone on its command line. |
| 0 | No. LIB file. ED did not find the LIB file specified in an R command. |
| > | Buffer full. ED cannot put anymore characters in the memory buffer, or string specified in an F, N, or S command is too long. |
| E | Command aborted. A keystroke at the keyboard aborted command execution. |
| F | File error. Followed by either disk FULL or DIRECTORY FULL. |

The following examples show how to recover from common editing error conditions. For example

```
BREAK ">" AT A
```

means that ED filled the memory buffer before completing the execution of an A command. When this occurs, the character pointer is at the end of the buffer and no editing is possible. Use the 0W command to write out half the buffer or use an O or H command and reedit the file.

```
BREAK "#" AT F
```

means that ED reached the end of the memory buffer without matching the string in an F command. At this point, the character pointer is at the end of the buffer. Move the CP with a B or n: line number command to resume editing.

```
BREAK "F" AT F
DISK FULL
```

Use the 0X command to erase an unnecessary file on the disk or a B#Xd:buffer.sav command to write the contents of the memory buffer onto another disk.

```
BREAK "F" AT n
DIRECTORY FULL
```

Use the same commands described in the previous message to recover from this file error.

The following table defines the disk file error messages ED returns when it cannot read or write a file.

**Table 6-5. ED Diskette File Error Messages**

| *Message* | *Meaning* |
|---|---|
| `CP/M Error on d:` `BDOS Function = NN` | `Read/Only File` `File = FILENAME.TYP`<br>Disk d: has Read-Only attribute. This occurs if a different disk has been inserted in the drive since the last cold or warm boot. |
| `** FILE IS READ ONLY **` | The file specified in the command to invoke ED has the R/O attribute. ED can read the file so that the user can examine it, but ED cannot change a Read-Only file. |

*End of Section 6*

# Appendix A
# CP/M 3 Messages

Messages come from several different sources. CP/M 3 can display error messages when the Basic Disk Operating System (BDOS) returns an error code. CP/M 3 can also display messages when there are errors in command lines. Each utility supplied with CP/M 3 has its own set of messages. The following table lists CP/M 3 messages and utility messages. If you are running an application program, you might see messages other than those listed here. Check the application program's documentation for explanations of those messages.

The messages in Table A-1 might be preceded by ERROR:. Some of them might also be preceded or followed by the filespec of the file causing the error condition. Sometimes the input line is flagged with an up arrow ↑, to indicate the character that caused the error. In this case, the message, `Error at the ^`, precedes the appropriate error message. Some of the messages are followed by an additional line preceded by INPUT:, OPTION:, or DRIVE: followed by the applicable error message.

**Table A-1. CP/M 3 Messages**

| *Message* | *Meaning* |
|---|---|
| `Assign a password to this file.` | |
| | SET. A password mode has been selected for this file but no password has been assigned. |
| `Auxiliary device redirection not implemented.` | |
| | GET and PUT. AUXIN and AUXOUT cannot be redirected to a file. |
| `Bad character, re-enter` | |
| | GENCPM. The character entered was not a number. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Bad close.` | |
| | SAVE. An error occurred during the attempt to close the file, probably because the file is write-protected. |
| `Bad Logical Device Assignment;` | |
| | DEVICE. Only the following logical devices are valid: CONIN:, CONOUT:, AUXIN:, AUXOUT:, LST:. |
| `BAD PARAMETER` | |
| | PIP. You entered an illegal parameter in a PIP command. Retype the entry correctly. |
| `Bad password.` | |
| | RENAME. The password supplied by the user is incorrect. |
| `Bank one not allowed.` | |
| | GENCPM. Bank 1 cannot be defined as available during system generation. |
| `Baud rate cannot be set for this device.` | |
| | DEVICE. Only physical devices that have the SOFT-BAUD attribute can have their baud rates changed. To check the attributes of the physical device, type DEVICE physical-dev. |

**Table A-1. (continued)**

| *Message* | *Meaning* |
|---|---|
| `Break "x" at c` | ED. "x" is one of the symbols described below and c is the command letter being executed when the error occurred.<br>`#` Search failure. ED cannot find the string specified in an F, N, or S command.<br>`?` Unrecognized command letter c. ED does not recognize the indicated command letter, or an E, H, O, or Q command is not alone on its command line.<br>`O` The file specified in an R command cannot be found.<br>`>` Buffer full. ED cannot put any more characters in the memory buffer, or the string specified in an F, N, or S command is too long.<br>`E` Command aborted. A keystroke at the console aborted command execution.<br>`F` Disk or directory full. This error is followed by either the disk or directory full message. Refer to the recovery procedures listed under these messages. |
| `CANNOT CLOSE:`<br>`Cannot close file.`<br>`CANNOT CLOSE FILE.`<br>`CANNOT CLOSE DESTINATION FILE - filespec` | GENCOM, HEXCOM, LIB-80™, LINK-80, MAC, PIP, RMAC, SUBMIT. An output file cannot be closed. This can occur if the disk is removed before the program terminates. |
| `Cannot delete file.` | GENCOM. CP/M cannot delete a file. Check to see if the COM file is Read-Only or password-protected. |

Table A-1. (continued)

| *Message* | *Meaning* |
| --- | --- |
| `Cannot have both create and access time stamps.` | |
| | SET. CP/M 3 supports either create or access time stamps, but not both. |
| `Cannot label a drive with a file referenced.` | |
| | SET. SET does not allow mixing of files and drives. |
| `CANNOT OPEN SOURCE FILE` | |
| | HEXCOM. The HEX file is not on the specified drive(s). |
| `Cannot redirect from BIOS.` | |
| | GET, PUT. This message is displayed as a warning only if the system has an invalid BIOS. |
| `Cannot set both RO and RW.` | |
| | SET. A file cannot be set to both Read-Only and Read-Write. |
| `Cannot set both SYS and DIR.` | |
| | SET. A file cannot be set to both SYS and DIR. |
| `CAN'T DELETE TEMP FILE` | |
| | PIP. A temporary $$$ file already exists which is Read-Only. Use the SET command to change the attribute to Read-Write, then erase it. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `CHECKSUM ERROR.`<br>`Checksum error` | HEXCOM, PIP. A hex record checksum error was encountered. The hex record that produced the error must be corrected, probably by recreating the hex file. |
| `Close error.` | XREF. This message is preceded by the filename.XRF. The disk might have been removed before the program terminated. |
| `Close operation failed.` | COPYSYS. There was a problem in closing the file at the end of the file copy operation. |
| `Closing file HELP.DAT`<br>`Closing file HELP.HLP` | HELP. HELP encountered error while processing the HELP.DAT or the HELP.HLP file. |
| `COM file found and NULL option.` | GENCOM. The NULL option implies that no COM file is to be loaded, just the RSXs. |
| `.COM file required` | DIR, ERASE, RENAME, TYPE. Options in the built-in command line require support from a transient COM file that CP/M 3 cannot find on disk. |
| `COMMON ERROR:` | LINK-80. An undefined common block has been selected. |

Table A-1. (continued)

| *Message* | *Meaning* |
| --- | --- |
| `CORRECT ERROR, TYPE RETURN OR CTRL-Z` | PIP. A hex record checksum was encountered during the transfer of a hex file. The hex file with the checksum error should be corrected, probably by recreating the hex file. |
| `CPMLDR error: failed to open CPM3.SYS` | CPMLDR. The system file CPM3.SYS is missing. |
| `CPMLDR error: failed to read CPM3.SYS` | CPMLDR. An error occurred while reading CPM3.SYS. |
| `CP/M Error on d: Disk I/O`<br>`BDOS Function = xx File = filespec` | CP/M displays the preceding message if the disk is defective or improperly formatted (wrong density). |
| `CP/M Error on d: Invalid Drive`<br>`BDOS Function = xx File = filespec` | CP/M 3 displays the preceding message when there is no disk in the drive, the drive latch is open, or the power is off. It also displays the message when the specified drive is not in the system. |
| `CP/M Error on d: Read/Only Disk`<br>`BDOS Function = xx File = filespec` | CP/M 3 does not allow you to erase, rename, update, or set attributes of a file residing in a Read-Only drive. Use the SET command to set the drive attribute to Read-Write. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `CP/M Error on d: Read/Only File`<br>`BDOS Function = xx File = filespec` | |
| | CP/M 3 does not allow you to erase, rename, update, or set attributes of a file that is Read-Only. Use the SET command to set the file attribute to Read-Write. |
| `Date and Time Stamping Inactive.` | |
| | DIR. The DATE option was specified, but the disk directory has not been initialized with date/time stamping. |
| `DESTINATION IS R/O, DELETE (Y/N)?` | |
| | PIP. The destination file specified in a PIP command already exists and it is Read-Only. If you type Y, the destination file is deleted before the file copy is done. If you type N, PIP displays the message **NOT DELETED** and aborts the copy operation. |
| `Device Reassignment Not Supported.`<br>`Enter new assignment or hit RETURN.` | |
| | DEVICE. A device assignment is invalid. |
| `Directory already re-formatted.` | |
| | INITDIR. The directory already has date/time stamping. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Directory full`<br>`DIRECTORY FULL` | ED. There is not enough directory space for the file being written to the destination disk. You can use the 0Xfilespec command to erase any unnecessary files on the disk without leaving the editor.<br>SUBMIT. There is not enough directory space on the temporary file drive to write the temporary file used for processing SUBMIT files. Use the SETDEF command to determine which drive is the temporary file drive. Use the ERASE command to erase unnecessary files or set the temporary file drive to a different drive and retry.<br>LIB-80, LINK-80. There is no directory space for the output or intermediate files. Use the ERASE command to remove unnecessary files.<br>GENCPM. There is no directory space for CPM3.SYS.<br>HEXCOM. There is no directory space for the output COM file. |
| `Directory needs to be reformatted for date/time stamps.` | SET. A date/time option was specified, but the directory has not been initialized for date/time stamping. Use the INITDIR command to initialize the directory for date/time stamping. |
| `DISK FULL` | ED. There is not enough disk space for the output file. This error can occur on the E, H, W, or X commands. If it occurs with X command, you can repeat the command prefixing the filename with a different drive. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `DISK READ`<br>`DISK READ ERROR:`<br>`Disk read error: filespec`<br>`DISK READ ERROR - filespec` | GENCPM, HEXCOM, LIB-80, LINK-80, PIP. The disk file specified cannot be read. |
| `DISK WRITE.`<br>`Disk Write Error`<br>`DISK WRITE ERROR:`<br>`DISK WRITE ERROR - filespec` | HEXCOM, LIB-80, LINK-80, PIP, SUBMIT. A disk write operation cannot be successfully performed probably because the disk is full. Use the ERASE command to remove unnecessary files. |
| `Do you want another file? (Y/N)` | PUT. Enter Y to redirect output to an additional file. Otherwise, enter N. |
| `Drive defined twice in search path` | SETDEF. A drive can be specified only once in the search path order. |
| `Drive Read Only` | ERASE, RENAME. The specified file is on a Read-Only drive and cannot be erased or renamed. |
| `Drive specified has not been defined.` | GENCPM. The drive specified has not been defined yet. Buffer(s) have not been allocated for the drive. |

Table A-1. (continued)

| Message | Meaning |
|---|---|
| `Duplicate RSX in header. Replacing old by new. This file was not used.` | GENCOM. The specified RSX is already attached to the COM file. The old one is discarded. |
| `Duplicate input RSX.` | GENCOM. Two or more RSXs of the same name are specified. GENCOM uses only one of the RSXs. |
| `Equals (=) delimiter missing at line NN.` | GENCPM. The equal sign is missing in the specified line. |
| `END OF FILE, ^Z, ?` | PIP encountered an unexpected end-of-file during a HEX file transfer. |
| `End of line expected.` | DEVICE, GET, PUT, SETDEF. The command typed does not have any further parameters. An end-of-line was expected. Any further characters on the line were ignored. |
| `Error at end of line:` | DEVICE, GET, PUT, SETDEF. The error detected occurred at the end of the input line. |
| `Error on line nnnnn:` | SUBMIT. The SUBMIT program displays its messages in the preceding format, where nnnnn represents the line number of the SUBMIT file. Refer to the message following the line number for explanation of the error. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `FILE ERROR` | |
| | ED. Disk or directory is full, and ED cannot write anything more on the disk. This is a fatal error, so make sure there is enough space on the disk to hold a second copy of the file before invoking ED. |
| `File already exists; Delete it? (Y/N)`<br>`file already exists, delete (Y/N)?` | |
| | PUT. Enter Y to delete the file. Otherwise the program terminates.<br>RENAME. The above message is preceded by filespec. You have asked CP/M 3 to create or rename a file using a file specification that is already assigned to another file. Either delete the existing file or use another file specification. |
| `File cannot fit into GENCPM buffer: filename.SPR` | |
| | GENCPM. There is not enough memory to generate a system. |
| `File exists, erase it` | |
| | ED. The destination filename already exists when you are placing the destination file on a different disk than the source. It should be erased or another disk selected to receive the output file. |
| `FILE IS READ/ONLY`<br>`File is Read Only` | |
| | ED. The file specified in the command to invoke ED has the Read-Only attribute. ED can read the file so that you can examine it, but ED cannot change a Read-Only file.<br>PUT. The file specified to receive the output is a Read-Only file. |

**Table A-1. (continued)**

| *Message* | *Meaning* |
|---|---|
| `FILE NAME ERROR:` | LIB-80. The form of a source filename is invalid. |
| `File not found.`<br>`FILE NOT FOUND - filespec` | DUMP, ED, GENCOM, GET, PIP, SET. An input file that you have specified does not exist. Check that you have entered the correct drive specification or that you have the correct disk in the drive. |
| `First submitted file must be a COM file.` | GENCOM. A COM file is expected as the first file in the command tail. The only time GENCOM does not expect to see a COM file in the first position of the command tail is when the NULL option is specified. |
| `FIRST COMMON NOT LARGEST:` | LINK-80. A subsequent COMMON declaration is larger than the first COMMON declaration for the indicated block. Check that the files being linked are in the proper order, or that the modules in a library are in the proper order. |
| `HELP.DAT not on current drive.` | HELP. HELP cannot find HELP.DAT file to process. |
| `Illegal command tail.` | DIR. The command line has an invalid format or option. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Illegal Format Value.` | |
| | DIR. Only SIZE and FULL options can be used for display formats. |
| `Illegal Global/Local Drive Spec Mixing.` | |
| | DIR. Both a filespec with a drive specifier and the DRIVE option appears in the command. |
| `Illegal filename.` | |
| | SAVE. There is an error in the filespec on the command line. |
| `Illegal Option or Modifier.` | |
| | DIR. An invalid option or abbreviation was used. |
| `Illegal date/time specification.` | |
| | DATE. Date/time format is invalid. |
| `Incorrect file specification.` | |
| | RENAME. The format of the filespec is invalid. |
| `INDEX ERROR:` | |
| | LINK-80. The index of an IRL contains invalid information. |
| `Insufficient Memory`<br>`INSUFFICIENT MEMORY:` | |
| | GET, LINK-80, PUT, SUBMIT. There is not enough memory to allocate buffers, or there are too many levels of SUBMIT nesting. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Invalid ASCII character` | |
| | SUBMIT. The SUBMIT file contains an invalid character (0FFH). |
| `Invalid character at line NN.` | |
| | GENCPM. The character must be a number. |
| `Invalid command.` | |
| | GET and PUT. The string or substring typed in the command line was not recognized as a valid command in the context used. |
| `Invalid delimiter.` | |
| | DEVICE, GET, PUT, SETDEF. The delimiter, [ ] , = or space, — was not valid at the location used. For example, a [ was used where an = should have been used. |
| `INVALID DESTINATION:` | |
| | PIP. An invalid drive or device was specified. |
| `INVALID DIGIT - filespec` | |
| | PIP. An invalid hex digit has been encountered while reading a hex file. The hex file with the invalid hex digit should be corrected, probably by recreating the hex file. |
| `Invalid drive.` | |
| | SETDEF. The specified drive was not a valid drive. Drives recognized by SETDEF are * (default drive) and A to P. GENCPM, TYPE. Valid drives are A to P. |

Table A-1. (continued)

| *Message* | *Meaning* |
| --- | --- |
| `Invalid drive ignored at line NN.` | GENCPM. Valid drives are A to P. |
| `Invalid drive name (Use A, B, C, or D)` | COPYSYS, GENCPM. Only drives A, B, C and D are valid destination drives for system generation. |
| `Invalid File.`<br>`INVALID FILENAME`<br>`Invalid file name.`<br>`Invalid Filename.`<br>`Invalid file specification.` | ED, ERASE, GENCOM, GET, PIP, PUT, SET, SUBMIT, TYPE. The filename typed does not conform to the normal CP/M 3 file naming conventions. |
| `INVALID FORMAT` | PIP. The format of your PIP command is illegal. See the description of the PIP command. |
| `INVALID HEX DIGIT.` | HEXCOM. An invalid hex digit has been encountered while reading a hex file. The hex file with the invalid hex digit should be corrected by recreating the hex file. |
| `Invalid number.` | DEVICE. A number was expected but not found, or the number was out of range; numbers must be from 0 to 255. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Invalid option.` | |
| | DEVICE and GET. An option was expected and the string found was not a device option or was not valid in the context used.<br>SETDEF. The option typed in the command line is not a valid option. Valid options are DISPLAY, NO DISPLAY, NO PAGE, ORDER, PAGE, TEMPORARY. |
| `Invalid option or modifier.` | |
| | DIR, GET, PUT. The option typed is not a valid option. |
| `INVALID PARAMETER:` | |
| | MAC, RMAC. An invalid assembly parameter was found in the input line. The assembly parameters are printed at the console up to the point of the error. |
| `Invalid parameter variable at line NN.` | |
| | GENCPM. The parameter variable does not exist. Check spelling. |
| `INVALID PASSWORD`<br>`Invalid password or passwords not allowed.` | |
| | ED, PIP. The specified password is incorrect, or a password was specified, but the file is not password-protected. |
| `Invalid physical device.` | |
| | DEVICE. A physical device name was expected. The name found in the command string does not correspond to any physical device name in the system. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| INVALID REL FILE: | LINK-80. The file indicated contains an invalid bit pattern. Make sure that a REL or IRL file has been specified. |
| Invalid RSX type. | GENCOM. Filetype must be RSX. |
| Invalid SCB offset. | GENCOM. The specified SCB is out of range. The SCB offset range is 00H-64H. |
| INVALID SEPARATOR | PIP. You have placed an invalid character for a separator between two input filenames. |
| INVALID SOURCE | PIP. An invalid drive or device was specified. AUX and CON are the only valid devices. |
| Invalid type for ORDER option. | SETDEF. The type specified in the command line was not COM or SUB. |
| Invalid SYM file format | XREF. The filename.SYM file input to XREF is invalid. |
| INVALID USER NUMBER | PIP. You have specified a user number greater than 15. User numbers are in the range 0 to 15. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Invalid wildcard.` | RENAME. The filespec contained an invalid wildcard specification. |
| `Invalid wild card in the FCB name or type field.` | GENCOM. GENCOM does not allow wildcards in filespecs. |
| `LOAD ADDRESS LESS THAN 100.` | HEXCOM. The program origin is less than 100H. |
| `MAIN MODULE ERROR:` | LINK-80. A second main module was encountered. |
| `Make error` | XREF. There is not more directory space on the specified drive. |
| `Memory conflict - cannot trim segment.` | GENCPM. The defined memory segment overlaps another segment. |
| `Memory conflict - segment trimmed.` | GENCPM. The defined memory segment overlaps with other segments. |
| `MEMORY OVERFLOW:` | LINK-80. There is not enough memory to complete the link operation. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Minimum number of buffers is 1.` | GENCPM. The first drive must have at least one buffer defined. |
| `Missing Delimiter or Unrecognized Option.` | ERASE. The ERASE command line format is invalid. |
| `Missing parameter variable at line NN.` | GENCPM. The line is missing a variable name. |
| `Missing left parenthesis.` | GENCOM. The SCB option must be enclosed by a left parenthesis. |
| `Missing right parenthesis.` | GENCOM. The SCB option is not enclosed with a right parenthesis. |
| `Missing SCB value.` | GENCOM. The SCB option requires a value. |
| `More than four drives specified.` | SETDEF. More than four drives were specified for the drive search chain. |
| `MULTIPLE DEFINITION:` | LINK-80. The specified symbol is defined in more than one of the modules being linked. |

Table A-1. (continued)

| *Message* | *Meaning* |
| --- | --- |
| `n?` | USER. You specified a number greater than fifteen for a user area number. For example, if you type USER 18, the screen displays 18?. |
| `No directory label exists.` | SHOW. The LABEL option was requested but the disk has no label. |
| `No directory space`<br>`NO DIRECTORY SPACE - filespec` | COPYSYS, GENCOM, MAC, PIP, RMAC, AND SAVE. There is not enough directory space for the output file. Use the ERASE command to remove unnecessary files on the disk and try again. |
| `No disk space.` | SAVE. There is not enough space on the disk for the output file. Use the SHOW command to display the amount of disk space left and use the ERASE command to remove unnecessary files from the disk, or use another disk with more file space. |
| `No file`<br>`NO FILE:`<br>`NO FILE - filespec` | DIR, ERASE, LIB-80, LINK-80, PATCH, PIP, RENAME, TYPE. The specified file cannot be found in the specified drive(s). |
| `No HELP.HLP file on the default drive.` | HELP. The file HELP.HLP must be on the default drive. |

**Table A-1. (continued)**

| *Message* | *Meaning* |
|---|---|
| `NO INPUT FILE PRESENT ON DISK` | |
| | DUMP. The file you requested does not exist. |
| `No memory` | |
| | There is not enough memory available for loading the specified program. |
| `No modifier for this option.` | |
| | GENCOM. A modifier was specified but none is required. |
| `NO MODULE:` | |
| | LIB-80. The indicated module cannot be found. |
| `No more space in the header for RSXs or SCB initialization.` | |
| | GENCOM. The header has room for only 15 entries, or the combination of RSXs and SCBs exceed the maximum. |
| `No options specified.` | |
| | SET. Specify an option. |
| `No PRN file.` | |
| | XREF. The file filename.PRN is not present on the specified drive. |
| `No Records Exist` | |
| | DUMP. Only a directory entry exists for the file. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `No source file on disk.` | COPYSYS. The file CPM3.SYS is not on the disk specified. |
| `NO SOURCE FILE PRESENT:` | MAC, RMAC. The source file cannot be found on the specified drive. |
| `NO SPACE` | SAVE. There is no space in the directory for the file being written. |
| `No 'SUB' file found.` | SUBMIT. The SUB file typed in the command line cannot be found in the drive search process. |
| `No such file to rename.` | RENAME. The file to be renamed does not exist on the specified drive(s). |
| `No SYM file` | XREF. The file filename .SYM is not present on the specified drive. |
| `NON-SYSTEM FILE(S) EXIST` | DIRS. If nonsystem (DIR) files reside on the specified drive, DIRS displays this message. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| Not enough available memory.<br>Not Enough Memory<br>Not Enough Memory for Sort. | DIR, INITDIR. There is not enough memory for data or sort buffers. |
| Not enough room in directory. | INITDIR. There is not enough remaining directory space to allow for the date and time extension. |
| NOT FOUND | PIP. PIP cannot find the specified file. |
| Not renamed, filespec read only. | RENAME. The specified file cannot be renamed because it is Read-Only. |
| OPEN FILE NONRECOVERABLE | PIP. A disk has the wrong format or a bad sector. |
| Option only for drives. | SET. The specified option is not valid for files. |
| Option requires a file reference. | SET. The specified option requires a filespec. |
| Out of data space. | COPYSYS. The destination drive ran out of space during the transfer of the CPM3.SYS file. |

Table A-1. (continued)

| *Message* | *Meaning* |
| --- | --- |
| `Options not grouped together.` | DIR. Options can only be specified within one set of brackets. |
| `Output File Exists, Erase it.` | The output file specified must not already exist. |
| `OUTPUT FILE READ ERROR:` | MAC, RMAC. An output file cannot be written properly, probably because the disk is full. Use the ERASE command to delete unnecessary files from the disk. |
| `OVERLAPPING SEGMENTS:` | LINK-80. LINK-80 attempted to write a segment into memory already used by another segment. |
| `Page and nopage option selected.`<br>`No page in effect.` | SET. The preceding options are mutually exclusive. |
| `Parameter Error` | SUBMIT. Within the SUBMIT file of type SUB, valid parameters are $0 through $9. |
| `Password Error.` | DUMP, ERASE, GENCOM, TYPE. The password is incorrect. |
| `Physical Device Does Not Exist.` | DEVICE. The specified physical device is not defined in the system. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Possible incompatible disk format.` | |
| | COPYSYS. The system disk and the output disk have different formats. |
| `PROGRAM INPUT IGNORED.` | |
| | SUBMIT. This message is preceded by "WARNING". The SUBMIT file contains a line with <, and the program does not require additional input. |
| `PUT>` | |
| | PUT. This prompt occurs when a program requests input while running a PUT FILE [NO ECHO] command. |
| `PUT ERROR: FILE ERASED.` | |
| | PUT. The PUT output file was erased and could not be closed. |
| `QUIT NOT FOUND` | |
| | PIP. The string argument to a Q parameter was not found in your input file. |
| `Random Read` | |
| | SUBMIT. An error occurred when reading the temporary file used by the SUBMIT command. |
| `Read only.` | |
| | GENCOM, SET. The drive or file specified is write-protected. |
| `Read error` | |
| | TYPE. An error occurred when reading the file specified in the TYPE command. Check the disk and try again. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Reading file: filespec` | GENCPM. An error occurred while attempting to read the file specified by filespec. |
| `Reading file HELP.HLP`<br>`Reading HELP.HLP index` | HELP. An error occurred while reading HELP.HLP. Copy the HELP.HLP file from the system disk. |
| `RECORD TOO LONG` | PIP. A HEX record exceeds 80 characters in a file being copied with the [H] option. |
| `Requires CP/M 3.0 or higher.` | DATE, DEVICE, DIR, ERASE, GENCOM, HELP, INITDIR, PIP, SET, SETDEF, SHOW, RENAME, TYPE. This version of the utility must only be run under CP/M 3.0 or higher. |
| `R/O DISK` | PIP. The destination drive is set to Read-Only and PIP cannot write to it. |
| `R/O FILE` | PIP. The destination file is set to Read-Only and PIP cannot write to it. |
| `Sort Stack Overflow` | DIR. There is not enough memory available for the sort stack. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Source file is incomplete.` | GENCPM. GENCPM cannot use your CP/M 3 system source file. |
| `SOURCE FILE READ ERROR:` | MAC, RMAC. The source file cannot be read properly by MAC. |
| `SOURCE FILENAME ERROR:` | MAC, RMAC. The form of the source filename is invalid. |
| `START NOT FOUND` | PIP. The string argument to an S parameter cannot be found in the source file. |
| `Symbol Table overflow` | XREF. No space is available for an attempted symbol allocation. |
| `Symbol Table reference overflow` | XREF. No space is available for an attempted symbol reference allocation. |
| `SYNTAX ERROR:` | LIB. The LIB-80 command is not properly formed. |
| `Too many entries in Index Table.` `Not enough memory` | HELP. There is not enough memory available to hold the topic table while creating HELP.HLP. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| Topic:<br>xxxxxx<br>Not found. | HELP. The topic requested does not exist in the HELP.HLP file. HELP displays the topics available. |
| Total file size exceeds 64K. | GENCOM. The output file exceeds the maximum allowed. |
| Try 'PAGE' or 'NO PAGE' | TYPE. The only valid option is PAGE or NO PAGE. |
| Unable to allocate Data deblocking buffer space. | GENCPM. There is not enough space left in generated system to allocate a data deblocking buffer. |
| Unable to allocate Dir deblocking buffer space. | GENCPM. There is not enough space left in generated system to allocate a directory deblocking buffer. |
| Unable to allocate space for hash table. | GENCPM. There is not enough contiguous memory to allocate space for the hash table in the generated system. |
| Unable to close HELP.DAT.<br>Unable to close HELP.HLP. | HELP. An error occurred while closing file HELP.HLP or HELP.DAT. There might not be enough disk or directory space on the drive. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Unable to find file HELP.HLP.` | HELP. HELP requires HELP.HLP file to operate. Copy it to your default drive from your CP/M 3 system disk. |
| `Unable to Make HELP.DAT.`<br>`Unable to Make HELP.HLP.` | HELP. There is not enough space on the disk for HELP.HLP or HELP.DAT, or the files are Read-Only. |
| `Unable to open: filename.SPR` | GENCPM. The file specified cannot be found on the default drive. |
| `UNBALANCED MACRO LIBRARY.` | MAC, RMAC. A MACRO definition was started within a macro library, but the end of the file was found in the library before the balancing ENDM was encountered. |
| `UNDEFINED START SYMBOL:` | LINK-80. The symbol specified with the G switch is not defined in any of the modules being linked. |
| `UNDEFINED SYMBOLS:` | LINK-80. The symbols following this message are referenced but not defined in any of the modules being linked. |
| `UNEXPECTED END OF HEX FILE - filespec` | PIP. An end-of-file was encountered before a termination hex record. The hex file without a termination record should be corrected, probably by recreating the hex file. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| Unrecognized drive. | SHOW. The specified drive is not valid. Valid drives are A to P. |
| UNRECOGNIZED ITEM: | LINK-80. An unfamiliar bit pattern has been scanned and ignored by LINK-80. |
| Unrecognized input. | SHOW. The SHOW command line has an invalid format. |
| Unrecognized option. | GENCOM and SHOW. An option typed in the command line is not valid for the command. |
| USER ABORTED | PIP. You stopped a PIP operation by pressing CTRL-C. |
| VERIFY ERROR: - filespec | PIP. When copying with the V option, PIP found a difference when rereading the data just written and comparing it to the data in its memory buffer. |

Table A-1. (continued)

| *Message* | *Meaning* |
|---|---|
| `Write error` | XREF. This message is preceded by filename.XRF and indicates that no disk space is available, or no directory space exists on the specified drive. |
| `Write protected?` | COPYSYS. The drive or disk to which the system is to be written is Read-Only. |
| `Writing file: filespec` | GENCPM, HELP. An error occurred while attempting to write the file specified by filespec. |
| `Wrong Password.` | SET. The specified password is incorrect or invalid. |
| `Zero length segment not allowed.` | GENCPM. A memory segment cannot have zero length. |
| `0FFFFH is an invalid value in the DPH directory BCB address field.` | GENCPM. This value is allowed only in the DTABCB field. |
| `?` | SID. SID has encountered an error. |

*End of Appendix A*

# Appendix B
# ASCII and Hexadecimal Conversions

ASCII stands for American Standard Code for Information Interchange. The code contains 96 printing and 32 non-printing characters used to store data on a disk. Table B-1 defines ASCII symbols, then Table B-2 lists the ASCII and hexadecimal conversions. The table includes binary, decimal, hexadecimal, and ASCII conversions.

**Table B-1. ASCII Symbols**

| *Symbol* | *Meaning* | *Symbol* | *Meaning* |
|---|---|---|---|
| ACK | acknowledge | FS | file separator |
| BEL | bell | GS | group separator |
| BS | backspace | HT | horizontal tabulation |
| CAN | cancel | LF | line-feed |
| CR | carriage return | NAK | negative acknowledge |
| DC | device control | NUL | null |
| DEL | delete | RS | record separator |
| DLE | data link escape | SI | shift in |
| EM | end of medium | SO | shift out |
| ENQ | enquiry | SOH | start of heading |
| EOT | end of transmission | SP | space |
| ESC | escape | STX | start of text |
| ETB | end of transmission | SUB | substitute |
| ETX | end of text | SYN | synchronous idle |
| FF | form-feed | US | unit separator |
| | | VT | vertical tabulation |

Table B-2. ASCII Conversion Table

| *Binary* | *Decimal* | *Hexadecimal* | *ASCII* |
|---|---|---|---|
| 0000000 | 0 | 0 | NUL |
| 0000001 | 1 | 1 | SOH (CTRL-A) |
| 0000010 | 2 | 2 | STX (CTRL-B) |
| 0000011 | 3 | 3 | ETX (CTRL-C) |
| 0000100 | 4 | 4 | EOT (CTRL-D) |
| 0000101 | 5 | 5 | ENQ (CTRL-E) |
| 0000110 | 6 | 6 | ACK (CTRL-F) |
| 0000111 | 7 | 7 | BEL (CTRL-G) |
| 0001000 | 8 | 8 | BS (CTRL-H) |
| 0001001 | 9 | 9 | HT (CTRL-I) |
| 0001010 | 10 | A | LF (CTRL-J) |
| 0001011 | 11 | B | VT (CTRL-K) |
| 0001100 | 12 | C | FF (CTRL-L) |
| 0001101 | 13 | D | CR (CTRL-M) |
| 0001110 | 14 | E | SO (CTRL-N) |
| 0001111 | 15 | F | SI (CTRL-O) |
| 0010000 | 16 | 10 | DLE (CTRL-P) |
| 0010001 | 17 | 11 | DC1 (CTRL-Q) |
| 0010010 | 18 | 12 | DC2 (CTRL-R) |
| 0010011 | 19 | 13 | DC3 (CTRL-S) |
| 0010100 | 20 | 14 | DC4 (CTRL-T) |
| 0010101 | 21 | 15 | NAK (CTRL-U) |
| 0010110 | 22 | 16 | SYN (CTRL-V) |
| 0010111 | 23 | 17 | ETB (CTRL-W) |
| 0011000 | 24 | 18 | CAN (CTRL-X) |
| 0011001 | 25 | 19 | EM (CTRL-Y) |
| 0011010 | 26 | 1A | SUB (CTRL-Z) |
| 0011011 | 27 | 1B | ESC (CTRL-[) |
| 0011100 | 28 | 1C | FS (CTRL-\) |
| 0011101 | 29 | 1D | GS (CTRL-]) |
| 0011110 | 30 | 1E | RS (CTRL-^) |
| 0011111 | 31 | 1F | US (CTRL-_) |
| 0100000 | 32 | 20 | (SPACE) |
| 0100001 | 33 | 21 | ! |
| 0100010 | 34 | 22 | " |
| 0100011 | 35 | 23 | # |
| 0100100 | 36 | 24 | $ |
| 0100101 | 37 | 25 | % |

**Table B-2. (continued)**

| *Binary* | *Decimal* | *Hexadecimal* | *ASCII* |
|---|---|---|---|
| 0100110 | 38 | 26 | & |
| 0100111 | 39 | 27 | ' |
| 0101000 | 40 | 28 | ( |
| 0101001 | 41 | 29 | ) |
| 0101010 | 42 | 2A | * |
| 0101011 | 43 | 2B | + |
| 0101100 | 44 | 2C | , |
| 0101101 | 45 | 2D | - |
| 0101110 | 46 | 2E | . |
| 0101111 | 47 | 2F | / |
| 0110000 | 48 | 30 | 0 |
| 0110001 | 49 | 31 | 1 |
| 0110010 | 50 | 32 | 2 |
| 0110011 | 51 | 33 | 3 |
| 0110100 | 52 | 34 | 4 |
| 0110101 | 53 | 35 | 5 |
| 0110110 | 54 | 36 | 6 |
| 0110111 | 55 | 37 | 7 |
| 0111000 | 56 | 38 | 8 |
| 0111001 | 57 | 39 | 9 |
| 0111010 | 58 | 3A | : |
| 0111011 | 59 | 3B | ; |
| 0111100 | 60 | 3C | < |
| 0111101 | 61 | 3D | = |
| 0111110 | 62 | 3E | > |
| 0111111 | 63 | 3F | ? |
| 1000000 | 64 | 40 | @ |
| 1000001 | 65 | 41 | A |
| 1000010 | 66 | 42 | B |
| 1000011 | 67 | 43 | C |
| 1000100 | 68 | 44 | D |
| 1000101 | 69 | 45 | E |
| 1000110 | 70 | 46 | F |
| 1000111 | 71 | 47 | G |
| 1001000 | 72 | 48 | H |
| 1001001 | 73 | 49 | I |
| 1001010 | 74 | 4A | J |
| 1001011 | 75 | 4B | K |

Table B-2. (continued)

| *Binary* | *Decimal* | *Hexadecimal* | *ASCII* |
|---|---|---|---|
| 1001100 | 76 | 4C | L |
| 1001101 | 77 | 4D | M |
| 1001110 | 78 | 4E | N |
| 1001111 | 79 | 4F | O |
| 1010000 | 80 | 50 | P |
| 1010001 | 81 | 51 | Q |
| 1010010 | 82 | 52 | R |
| 1010011 | 83 | 53 | S |
| 1010100 | 84 | 54 | T |
| 1010101 | 85 | 55 | U |
| 1010110 | 86 | 56 | V |
| 1010111 | 87 | 57 | W |
| 1011000 | 88 | 58 | X |
| 1011001 | 89 | 59 | Y |
| 1011010 | 90 | 5A | Z |
| 1011011 | 91 | 5B | [ |
| 1011100 | 92 | 5C | \ |
| 1011101 | 93 | 5D | ] |
| 1011110 | 94 | 5E | ^ |
| 1011111 | 95 | 5F | < |
| 1100000 | 96 | 60 | ' |
| 1100001 | 97 | 61 | a |
| 1100010 | 98 | 62 | b |
| 1100011 | 99 | 63 | c |
| 1100100 | 100 | 64 | d |
| 1100101 | 101 | 65 | e |
| 1100110 | 102 | 66 | f |
| 1100111 | 103 | 67 | g |
| 1101000 | 104 | 68 | h |
| 1101001 | 105 | 69 | i |
| 1101010 | 106 | 6A | j |
| 1101011 | 107 | 6B | k |
| 1101100 | 108 | 6C | l |
| 1101101 | 109 | 6D | m |
| 1101110 | 110 | 6E | n |
| 1101111 | 111 | 6F | o |
| 1110000 | 112 | 70 | p |
| 1110001 | 113 | 71 | q |

**Table B-2. (continued)**

| *Binary* | *Decimal* | *Hexadecimal* | *ASCII* |
|---|---|---|---|
| 1110010 | 114 | 72 | r |
| 1110011 | 115 | 73 | s |
| 1110100 | 116 | 74 | t |
| 1110101 | 117 | 75 | u |
| 1110110 | 118 | 76 | v |
| 1110111 | 119 | 77 | w |
| 1111000 | 120 | 78 | x |
| 1111001 | 121 | 79 | y |
| 1111010 | 122 | 7A | z |
| 1111011 | 123 | 7B | { |
| 1111100 | 124 | 7C | \| |
| 1111101 | 125 | 7D | } |
| 1111110 | 126 | 7E | ~ |
| 1111111 | 127 | 7F | DEL |

*End of Appendix B*

# Appendix C
# Filetypes

CP/M 3 identifies every file by a unique file specification, which consists of a drive specification, a filename, a filetype, and an optional password. The filetype is an optional three-character ending separated from the filename by a period. The filetype generally indicates a special kind of file. The following table lists common filetypes and their meanings.

**Table C-1. Common Filetypes**

| *Type* | *Meaning* |
|---|---|
| ASM | Assembly language source file; the CP/M 3 assemblers assemble or translate a type ASM file into machine language. |
| BAK | Back-up file created by text editor; the editor renames the source file with this filetype to indicate that the original file has been processed. The original file stays on disk as the back-up file, so you can refer to it. |
| BAS | CBASIC program source file. |
| COM | 8080 executable file. |
| ERL | Pascal/MT+ ™ relocatable file. |
| HEX | Program file in hexadecimal format. |
| INT | CBASIC program intermediate language file. |
| IRL | Indexed REL file produced by LIB. |
| LIB | Used by MAC and RMAC for macro libraries. The ED R command reads files of type LIB. The ED X command writes files of type LIB. Printable file displayable on console or printer. |

**Table C-1. (continued)**

| *Type* | *Meaning* |
|---|---|
| OVL | Program overlay file. PL/I-80 compiler overlays files; you can create overlay files with LINK-80. |
| PAS | Pascal/MT+ source program filetype. |
| PLI | PL/I-80 source program filetype. |
| PRL | Page Relocatable file; a file that does not require an absolute segment. It can be relocated in any Page Boundary (256 Bytes). |
| PRN | Printable file displayable on console or printer. |
| REL | Relocatable file produced by RMAC and PL/I-80 that can be linked by LINK-80. |
| SPR | System Page Relocatable file; system files required to generate CP/M 3, such as BNKBDOS.SPR, BDOS.SPR, BIOS.SPR, and RESBDOS.SPR. |
| SUB | Filetype required for submit file containing one or more CP/M 3 commands. The SUBMIT program executes commands in files of type SUB, providing a batch execution mode for CP/M 3. |
| SYM | Symbol table file. MAC, RMAC, and LINK-80 output files of type SYM. SID and ZSID read files of type SYM. |
| SYS | System file for CP/M 3. |
| TEX | Source file for TEX-80™, the Digital Research text formatter. |
| TOK | Pascal/MT+ intermediate language file. |
| XRF | Cross-reference file produced by XREF. |
| $$$ | Temporary file. |

*End of Appendix C*

# Appendix D
# CP/M 3 Control Character Summary

**Table D-1. Nonbanked CP/M 3 Control Characters**

| *Character* | *Meaning* |
|---|---|
| CTRL-C | Terminates the executing program and redisplays the system prompt, provided the cursor is at the beginning of the command line. Also, if you halt scrolling with CTRL-S, you can terminate the program with a CTRL-C. |
| CTRL-E | Forces a physical carriage return but does not send the command line to CP/M 3. Moves the cursor to the beginning of the next line without erasing your previous input. |
| CTRL-H | Deletes a character and moves the cursor left one character position. |
| CTRL-I | Moves the cursor to the next tab stop. Tab stops are automatically set at each eighth column. Has the same effect as pressing the TAB key. |
| CTRL-J | Sends the command line to CP/M 3 and returns the cursor to the left of the current line. Has the same effect as a RETURN or a CTRL-M. |
| CTRL-M | Sends the command line to CP/M 3 and returns the cursor to the left of the current line. Has the same effect as a RETURN or a CTRL-J. |
| CTRL-P | Echoes all console activity to the printer. The first time you type CTRL-P, CP/M 3 rings a bell at your console. You can use CTRL-P after you halt scrolling with CTRL-S. A second CTRL-P ends printer echo; no bell rings. CTRL-P has no effect if your system does not include a printer. |

Table D-1. (continued)

| *Character* | *Meaning* |
|---|---|
| CTRL-R | Places a # at the current cursor location, moves the cursor to the next line, and displays any partial command you typed so far. |
| CTRL-S | Stops screen scrolling. If a display scrolls by too fast for you to read it, type CTRL-S. CTRL-Q restarts screen scrolling. |
| CTRL-U | Discards all the characters in the command line, places a # at the current cursor position, and moves the cursor to the next command line. |
| CTRL-X | Discards all the characters in the command line, and moves the cursor to the beginning of the current line. |

Table D-2. Banked CP/M 3 Line-editing Control Characters

| *Character* | *Meaning* |
|---|---|
| CTRL-A | Moves the cursor one character to the left. |
| CTRL-B | Moves the cursor to the beginning of the command line without having any effect on the contents of the line. If the cursor is at the beginning, CTRL-B moves it to the end of the line. |
| CTRL-C | Terminates the executing program and redisplays the system prompt, provided the cursor is at the beginning of the command line. Also, if you halt scrolling with CTRL-S, you can terminate the program with a CTRL-C. |
| CTRL-E | Forces a physical carriage return but does not send the command line to CP/M 3. Moves the cursor to the beginning of the next line without erasing the previous input. |
| CTRL-F | Moves the cursor one character to the right. |
| CTRL-G | Deletes the character indicated by the cursor. The cursor does not move. |
| CTRL-H | Deletes a character and moves the cursor left one character position. |
| CTRL-I | Moves the cursor to the next tab stop. Tab stops are automatically set at each eighth column. Has the same effect as pressing the TAB key. |
| CTRL-J | Sends the command line to CP/M 3 and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or a CTRL-M keystroke. |
| CTRL-K | Deletes to the end of the line from the cursor. |
| CTRL-M | Sends the command line to CP/M 3 and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or a CTRL-J keystroke. |

**Table D-2. (continued)**

| *Character* | *Meaning* |
|---|---|
| CTRL-P | Echoes all console activity to the printer. The first time you type CTRL-P, CP/M 3 rings a bell at your console. You can use CTRL-P after you halt scrolling with CTRL-S. A second CTRL-P ends printer echo; no bell rings. CTRL-P has no effect if your system does not include a printer. |
| CTRL-R | Retypes the command line. Places a # at the current cursor location, moves the cursor to the next line, and retypes any partial command you typed so far. |
| CTRL-S | Stops screen scrolling. If a display scrolls by too fast for you to read it, type CTRL-S. CTRL-Q restarts screen scrolling. |
| CTRL-U | Discards all the characters in the command line, places a # at the current cursor position, and moves the cursor to the next line. However, you can use a CTRL-W to recall any characters that were to the left of the cursor when you pressed CTRL-U. |
| CTRL-W | Recalls and displays previously entered command line both at the operating system level and in executing programs, if the CTRL-W is the first character entered after the prompt. CTRL-J, CTRL-M, CTRL-U, and RETURN define the command line you can recall. If the command line contains characters, CTRL-W moves the cursor to the end of the command line. If you press RETURN, CP/M 3 executes the recalled command. |
| CTRL-X | Discards all the characters left of the cursor and moves the cursor to the beginning of the current line. CTRL-X saves any characters right of the cursor. |

*End of Appendix D*

# Appendix E
# User's Glossary

**ambiguous filename:** Filename that contains either of the CP/M 3 wildcard characters, ? or *, in the primary filename or the filetype or both. When you use wildcard characters, you create an ambiguous filespec and can easily reference more than one CP/M 3 file. See Section 2 of this manual.

**applications program:** Program that solves a specific problem. Typical applications programs are business accounting packages, word processing (editing) programs, and mailing list programs.

**argument:** Symbol indicating a place into which you can substitute a number, letter, or name to give an appropriate meaning to a command line.

**ASCII:** The American Standard Code for Information Interchange is a standard code for representation of numbers, letters, and symbols. An ASCII text file is a file that can be intelligibly displayed on the video screen or printed on paper. See Appendix B.

**attribute:** File characteristic that can be set to on or off.

**back-up:** Copy of a disk or file made for safe keeping, or the creation of the back-up disk or file.

**bit:** Switch in memory that can be set to on (1) or off (0). Bits are grouped into bytes.

**block:** Area of disk.

**bootstrap:** Process of loading an operating system into memory. Bootstrap procedures vary from system to system. The boot for an operating system must be customized for the memory size and hardware environment that the operating system manages. Typically, the boot is loaded automatically and executed at power up or when the computer is reset. Sometimes called a "cold start."

**buffer:** Area of memory that temporarily stores data during the transfer of information.

**built-in commands:** Commands that permanently reside in memory. They respond quickly because they are not accessed from a disk.

**byte:** Unit of memory or disk storage containing eight bits.

**character string:** Any combination of letters, numbers, or special characters on your keyboard.

**command:** Elements of a CP/M 3 command line. In general, a CP/M 3 command has three parts: the command keyword, the command tail, and a carriage return keystroke.

**command file:** Series of coded machine executable instructions stored on disk as a program file, invoked in CP/M 3 by typing the command keyword next to the system prompt on the console. CP/M 3 command files generally have a filetype of COM. Files are either command files or data files. Same as a command program.

**command keyword:** Name that identifies an CP/M 3 command, usually the primary filename of a file of type COM, or a built-in command. The command keyword precedes the command tail and the carriage return in the command line.

**command syntax:** Statement that defines the correct way to enter a command. The correct structure generally includes the command keyword, the command tail, and a carriage return. A syntax line usually contains symbols that you should replace with actual values when you enter the command.

**command tail:** Part of a command that follows the command keyword in the command line. The command tail can include a drive specification, a filename and/or filetype, and options or parameters, but cannot exceed 128 characters. Some commands do not require a command tail.

**concatenate:** Term that describes one of PIP's operations that combines two or more separate files into one new file in the specified sequence.

**console:** Primary input/output device. The console consists of a listing device such as a screen and a keyboard through which the user communicates with the operating system or applications program.

**control character:** Nonprinting character combination that sends a simple command to CP/M 3. Some control characters perform line editing functions. To enter a control character, hold down the CTRL key on your terminal and strike the character key specified. See Appendix D.

**cursor:** One-character symbol that can appear anywhere on the console screen. The cursor indicates the position where the next keystroke at the console will have an effect.

**data file:** Nonexecutable collection of similar information that generally requires a command file to manipulate it.

**default:** Currently selected disk drive and/or user number. Any command that does not specify a disk drive or a user number references the default disk drive and user number. When CP/M 3 is first invoked, the default disk drive is drive A, and the default user number is 0, until changed with the USER command.

**delimiter:** Special characters that separate different items in a command line. For example, in CP/M 3, a colon separates the drive spec from the filename. A period separates the filename from the filetype. Brackets separate any options from their command or filespec. Commas separate one item in an option list from another. All of the preceding special characters are delimiters.

**directory:** Portion of a disk that contains descriptions of each file on the disk. In response to the DIR command, CP/M 3 displays the filenames stored in the directory.

**DIR attribute:** File attribute. A file with the DIR attribute can be displayed by a DIR command. The file can be accessed from the default user number only.

**disk, diskette:** Magnetic media used to store information. Programs and data are recorded on the disk in the same way that music is recorded on a cassette tape. The term diskette refers to smaller capacity removable floppy diskettes. Disk can refer to a diskette, a removable cartridge disk, or a fixed hard disk.

**disk drive:** Peripheral device that reads and writes on hard or floppy disks. CP/M 3 assigns a letter to each drive under its control. For example, CP/M 3 can refer to the drives in a four-drive system as A, B, C, and D.

**editor:** Utility program that creates and modifies text files. An editor can be used for creation of documents or creation of code for computer programs. The CP/M 3 editor is invoked by typing the command ED next to the system prompt on the console. (See ED in Section 6 of this manual).

**executable:** Ready to be run by the computer. Executable code is a series of instructions that can be carried out by the computer. For example, the computer cannot execute names and addresses, but it can execute a program that prints all those names and addresses on mailing labels.

**execute a program:** Start a program executing. When a program is running, the computer is executing a sequence of instructions.

**FCB:** See **File Control Block.**

**file:** Collection of characters, instructions or data stored on a disk. The user can create files on a disk.

**File Control Block:** Structure used for accessing files on disk. Contains the drive, filename, filetype and other information describing a file to be accessed or created on the disk.

**filename:** Name assigned to a file. A filename can include a primary filename of 1-8 characters and a filetype of 0-3 characters. A period separates the primary filename from the filetype.

**file specification:** Unique file identifier. A complete CP/M 3 file specification includes a disk drive specification followed by a colon (d:), a primary filename of 1 to 8 characters, a period, and a filetype of 0 to 3 characters. For example, b:example.tex is a complete CP/M 3 file specification.

**filetype:** Extension to a filename. A filetype can be from 0 to 3 characters and must be separated from the primary filename by a period. A filetype can tell something about the file. Certain programs require that files to be processed have certain filetypes (see Appendix C).

**floppy disk:** Flexible magnetic disk used to store information. Floppy disks come in 5 1/4- and 8-inch diameters.

**hard disk:** Rigid, platter-like, magnetic disk sealed in a container. A hard disk stores more information than a floppy disk.

**hardware:** Physical components of a computer.

**hex file:** ASCII-printable representation of a command (machine language) file.

**hexadecimal notation:** Notation for the base 16 number system using the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F to represent the sixteen digits. Machine code is often converted to hexadecimal notation because it can be easily represented by ASCII characters and therefore printed on the console screen or on paper (see Appendix B).

**input:** Data going into the system, usually from an operator typing at the terminal or by a program reading from the disk.

**interface:** Object that allows two independent systems to communicate with each other, as an interface between hardware and software in a microcomputer.

**I/O:** Abbreviation for input/output.

**keyword:** See **command keyword.**

**kilobyte:** 1024 bytes denoted as 1K. 32 kilobytes equal 32K. 1024 kilobytes equal one megabyte, or over one million bytes.

**list device:** Device such as a printer onto which data can be listed or printed.

**logical:** Representation of something that might or might not be the same in its actual physical form. For example, a hard disk can occupy one physical drive, and yet you can divide the available storage on it to appear to the user as if it were in several different drives. These apparent drives are the logical drives.

**megabyte:** Over one million bytes; 1024 kilobytes. See **byte** and **kilobyte.**

**microprocessor:** Silicon chip that is the Central Processing Unit (CPU) of the microcomputer.

**operating system:** Collection of programs that supervises the running of other programs and the management of computer resources. An operating system provides an orderly input/output environment between the computer and its peripheral devices.

**option:** One of many parameters that can be part of a command tail. Use options to specifiy additional conditions for a command's execution.

**output:** Data that the system sends to the console or disk.

**parameter:** Value in the command tail that provides additional information for the command. Technically, a parameter is a required element of a command.

**peripheral devices:** Devices external to the CPU. For example, terminals, printers, and disk drives are common peripheral devices that are not part of the processor, but are used in conjunction with it.

**physical:** Actual hardware of a computer. The physical environment varies from computer to computer.

**primary filename:** First 8 characters of a filename. The primary filename is a unique name that helps the user identify the file contents. A primary filename contains 1 to 8 characters and can include any letter or number and some special characters. The primary filename follows the optional drive specification and precedes the optional filetype.

**program:** Series of specially coded instructions that performs specific tasks when executed by a computer.

**prompt:** Characters displayed on the screen to help the user decide what the next appropriate action is. A system prompt is a special prompt displayed by the operating system. The system prompt indicates to the user that the operating system is ready to accept input. The CP/M 3 system prompt is an alphabetic character followed by an angle bracket. The alphabetic character indicates the default drive. Some applications programs have their own special system prompts.

**Read-Only:** Attribute that can be assigned to a disk file or a disk drive. When assigned to a file, the Read-Only attribute allows you to read from that file but not change it. When assigned to a drive, the Read-Only attribute allows you to read any file on the disk, but prevents you from adding a new file, erasing or changing a file, renaming a file, or writing on the disk. The SET command can set a file or a drive to Read-Only. Every file and drive is either Read-Only or Read-Write. The default setting for drives and files is Read-Write, but an error in resetting the disk or changing media automatically sets the drive to Read-Only until the error is corrected. Files and disk drives can be set to either Read-Only or Read-Write.

**Read-Write:** Attribute that can be assigned to a disk file or a disk drive. The Read-Write attribute allows you to read from and write to a specific Read-Write file or to any file on a disk that is in a drive set to Read-Write. A file or drive can be set to either Read-Only or Read-Write.

**record:** Collection of data. A file consists of one or more records stored on disk. An CP/M 3 record is 128 bytes long.

**RO:** See **Read-Only.**

**RW:** See **Read-Write.**

**sector:** Portion of a disk track. There are a specified number of sectors on each track.

**software:** Specially coded programs that transmit machine-readable instructions to the computer, as opposed to hardware, which is the actual physical components of a computer.

**source file:** ASCII text file that is an input file for a processing program, such as an editor, text formatter, or assembler.

**string:** See **character string**

**syntax:** Format for entering a given command.

**system attribute:** File attribute. You can give a file the system attribute by using the SYS option in the SET command. A file with the SYS attribute is not displayed in response to a DIR command; you must use DIRS (see Section 5). If you give a file with user number 0 the SYS attribute, you can read and execute that file from any user number on the same drive. Use this feature to make your commonly used programs available under any user number.

**system prompt:** Symbol displayed by the operating system indicating that the system is ready to receive input. See **prompt.**

**terminal:** See **console.**

**track:** Concentric rings dividing a disk. There are 77 tracks on a typical eight-inch floppy disk.

**turn-key application:** Application designed for the noncomputer-oriented user. For example, a typical turn-key application is designed so that the operator needs only to turn on the computer, insert the proper program disk, and select the desired procedure from a selection of functions (menu) displayed on the screen.

**upward-compatible:** Term meaning that a program created for the previously released operating system (or compiler, etc.) runs under the newly released version of the same operating system.

**user number:** Number from 0 to 15 assigned to a file when it is created. User numbers can organize files into sixteen file groups.

**utility:** Tool. Program that enables the user to perform certain operations, such as copying files, erasing files, and editing files. Utilities are created for the convenience of programmers and users.

**wildcard characters:** Special characters that give CP/M 3 a pattern to match when it searches the directory for a file. CP/M 3 recognizes two wildcard characters, ? and *. The ? can be substituted for any single character in a filespec, and the * can be substituted for the primary filename or the filetype or both. By placing wildcard characters in a filespec, you create an ambiguous filespec and can quickly reference one or more files.

*End of Appendix E*

# Index

## B



## C

## D

**X**



**Y**



**Z**

# NOTES

## NOTES

## NOTES

DIGITAL RESEARCH®

# CP/M Plus™

Operating System

# Command Summary

NOTICE TO USER

From time to time changes are made in the filenames and in the files actually included on the distribution disk. This manual should not be construed as a representation or warranty that such files or facilities exist on the distribution disk or as part of the materials and programs distributed. Most distribution disks include a "README.DOC" file. This file explains variations from the manual which do constitute modification of the manual and the items included therewith. Be sure to read this file before using the software.



The CP/M Plus Operating System Command Summary was prepared using the Digital Research TEX™ Text Formatter and printed in the United States of America.

********************************
* First Edition: March 1984 *
********************************

# Table of Contents

# Table of Contents
(continued)

# HOW TO ENTER A CP/M PLUS COMMAND

To give CP/M Plus™ a command, type a complete command line following the CP/M Plus system prompt, A>. A CP/M Plus command line consists of a command, an optional command tail, and a RETURN or ENTER keystroke. The command is the name of a program to run. An optional command tail can consist of a drive letter followed by a colon, one or more file names, and some options. To complete the command you must press the RETURN or ENTER key.

A>**COMMAND** <RET>

# CP/M PLUS FILE SPECIFICATIONS

CP/M Plus identifies every file by its complete name or file specification. A file specification is any valid combination of the drive specification, filename, filetype, and password, all separated by their appropriate delimiters. A drive letter must be followed by a colon. A filetype must be preceded by a period. A password must be preceded by a semicolon. The term filespec is an abbreviation for file specification.

This summary uses the following symbols to designate the parts of a filespec.

| Symbol | Meaning |
|---|---|
| **d:** | Represents the optional drive specification, which can be any single alphabetic character in the range A through P, followed by a colon. |
| **filename** | Represents the required filename, which can be from 1 to 8 alphanumeric characters. |

| Symbol | Meaning |
|---|---|
| **.typ** | Represents the optional filetype, which can be from 0 to 3 alphanumeric characters; separated from the filename by a period. A period does not precede the filetype when the filetype is named alone in the text of this command summary. |
| **;password** | The optional password, which can be from 0 to 8 alphanumeric characters; separated from the filetype by a semicolon. |

Valid combinations of the elements of a filespec are shown below:

- filename
- d:filename
- filename.typ
- d:filename.typ
- filename;password
- d:filename;password
- filename.typ;password
- d:filename.typ;password

## COMMAND SUMMARY CONVENTIONS

The command summary alphabetically lists each CP/M Plus command using the following special symbols to define command syntax:

| Symbol | Meaning |
|---|---|
| {} | optional item |
| \| | 'or' bar; separates choices when only one option can be used at a time |
| n | number |
| CTRL | control key |
| o | option or an option list |
| <RET> | RETURN or ENTER key |

| Symbol | Meaning |
|---|---|
| **RW** | Read/Write |
| **RO** | Read/Only |
| **SYS** | System attribute; file does not appear when directory is displayed by DIR command |
| **DIR** | Directory attribute; file appears in response to DIR command |
| ... | means the element can be repeated as many times as you want |
| * | wildcard; replaces all or part of a filename and/or filetype; must be last character in filename or filetype |
| ? | wildcard; replaces any single character in the same position of a filename and/or filetype |
| [] | type square brackets to enclose an option list |

Certain CP/M Plus commands can be modified by options added to the command or file specification. Options are enclosed in square brackets. The right-hand square bracket is optional. Only one or two characters of the option word are necessary to specify an option. Generally, options can be grouped together, separated by commas or spaces in the square brackets. This does not apply to options that contradict each other. Sometimes the whole command tail is optional.

# CONTROL CHARACTERS

The following list presents CP/M Plus control characters and their functions.

| Control Character | Function |
|---|---|
| CTRL-A | Moves cursor one character to the left. Works only if your computer has bank-switched memory. |
| CTRL-B | Moves cursor from beginning to end of command line and back without affecting command. |

## CONTROL CHARACTERS (continued)

| Control Character | Function |
|---|---|
| CTRL-C | Stops executing program when entered at the system prompt or after CTRL-S. |
| CTRL-E | Forces a physical return without sending command to CP/M Plus. |
| CTRL-F | Moves cursor one character to the right. Banked system only. |
| CTRL-G | Deletes character at current cursor position if in the middle of a line. Banked system only. |
| CTRL-H | Delete character to the left of cursor. |
| CTRL-I | Same as the TAB key. |
| CTRL-J | Moves cursor to the left of the command line and sends command to CP/M Plus. Line feed; same effect as RETURN. |
| CTRL-K | Deletes character at cursor and all characters to the right. |
| CTRL-M | Same as RETURN. |
| CTRL-P | Echoes console output to the list device. |
| CTRL-Q | Restarts screen scrolling. |
| CTRL-R | Retypes the characters to the left of the cursor on a new line; updates the command line buffer. |
| CTRL-S | Stops screen scrolling. |
| CTRL-U | Updates the command line buffer to contain the characters to the left of the cursor; deletes current line. |
| CTRL-W | Recalls previous command line if current line is empty; otherwise moves cursor to end of line. CTRL-J,-M,-R,-U and RETURN update the command line buffer for recall with CTRL-W. Banked system only. |
| CTRL-X | Deletes all characters to the left of the cursor. |

# COPYSYS

**Syntax:**

COPYSYS

**Purpose:**

COPYSYS copies the CP/M Plus system from a CP/M Plus system diskette to another diskette. The new diskette must have the same format as the original system diskette.

**Example:**

A>**COPYSYS**

# DATE

**Syntax:**

```
DATE
DATE C
DATE CONTINUOUS
DATE time-specification
DATE SET
```

**Purpose:**

The DATE command lets you display and set the date and time of day.

**Examples:**

A>**DATE**

Displays the current date and time.

A>**DATE C**

Displays the date and time continuously.

A>**DATE 08/14/82 10:30:0**

Sets the date and time.

A>**DATE SET**

Prompts for date and time entries.

# DEVICE

---

**Syntax:**

```
DEVICE
DEVICE NAMES
DEVICE VALUES
DEVICE logical-dev {XON|NOXON|baud-rate}
DEVICE physical-dev {XON|NOXON|baud-rate}
DEVICE logical-dev=physical-dev {option}
          {,physical-dev {option},...}
DEVICE logical-dev = NULL
DEVICE CONSOLE [PAGE]
DEVICE CONSOLE [COLUMNS=n,LINES=n]
```

**Purpose:**

DEVICE displays current logical device assignments and physical device names.

**Examples:**

A>**DEVICE**

Displays the physical devices and current assignments of the logical devices in the system.

A>**DEVICE NAMES**

Lists the physical devices with a summary of the device characteristics.

A>**DEVICE VALUES**

Displays the current logical device assignments.

A>**DEVICE CRT**

Displays the attributes of the physical device CRT.

A>**DEVICE CON**

Displays the assignment of the logical device CON:

# DEVICE (continued)

**Purpose:**

DEVICE assigns logical devices to peripheral devices attached to the computer, and sets the communications protocol and speed of a peripheral device. Note that the physical devices available can differ from system to system. Consult your hardware manufacturer's documentation for valid device names. An explanation of the valid device options follows:

**DEVICE Options:**

| Option | Explanation |
|---|---|
| **XON** | Sets the device to the XON/XOFF communications protocol. |
| **NOXON** | Indicates no protocol; the computer sends data to the device whether or not the device is ready to receive it. |
| **baud-rate** | Is the speed of the device. The system accepts the following baud rates: |

| | | | |
|---|---|---|---|
| 50 | 75 | 110 | 134 |
| 150 | 300 | 600 | 1200 |
| 1800 | 2400 | 3600 | 4800 |
| 7200 | 9600 | 19200 | |

**Examples:**

A>**DEVICE CONOUT:=LPT,CRT**

Assigns the system console output (CONOUT:) to the printer (LPT:) and the screen (CRT:).

A>**DEVICE AUXIN:=CRT2 [XON,9600]**

Assigns the auxiliary logical input device (AUXIN:) to the physical device CRT2: using protocol XON/XOFF, and sets the transmission rate for the device at 9600.

A>**DEVICE LST:=NULL**

Disconnects the list output logical device (LST:).

A>**DEVICE LPT [XON,9600]**

Sets the XON/XOFF protocol for the physical device LPT: and sets the transmission speed at 9600.

# DEVICE (continued)

**Purpose:**

DEVICE displays or sets the current console screen size.

**Examples:**

A>**DEVICE CONSOLE [PAGE]**

Displays the current console page width in columns and length in lines.

A>**DEVICE CONSOLE [COLUMNS=40,LINES=16]**

Sets the screen size to 40 columns and 16 lines.

# DIR

**Syntax:**

```
DIR
DIR  d:
DIR  filespec

DIRS
DIRS d:
DIRS filespec

DIR  d: options
DIR  filespec,... filespec options
```

**Purpose:**

The DIR and DIRS commands display the names of files cataloged in the directory of an on-line disk. DIR lists the names of files in the current user number that have the Directory (DIR) attribute. DIRS lists the names of files in the current directory that have the system (SYS) attribute. DIR and DIRS accept the * and ? wildcards in the file specification.

The DIR command with options displays the names of files and the characteristics associated with the files.

DIR and DIRS are built-in utilities. DIR with options is a transient utility and must be loaded into memory from the disk.

**Examples:**

A>**DIR**

Displays all files in user 0 on drive A that have the Directory attribute.

A>**DIR B:**

Displays all DIR files in user 0 on drive B.

2A>**DIR C:ZIPPY.DAT**

Displays the name ZIPPY.DAT if the file is in user 2 on drive C.

4A>**DIR *.BAS**

Displays all DIR files with filetype BAS in user 4 on drive A.

## DIR (continued)

B3>**DIR X*.C?D**

Displays all DIR files in user 3 on drive B whose filename begins with the letter X, and whose three-character filetype contains the first character C and last character D.

A>**DIRS**

Displays all files for user 0 on drive A that have the system (SYS) attribute.

A>**DIRS *.COM**

Displays all SYS files with filetype COM on drive A in user 0. A command (COM) file in user 0 with the system attribute can be accessed from any user number on that drive, and from any drive in the search chain (see SETDEF).

**Purpose:**

The DIR command with options is an enhanced version of the DIR built-in command and displays your files in a variety of ways. DIR can search for files on any or all drives, for any or all user numbers. One or two letters is sufficient to identify an option. You need not type the right square bracket.

**DIR Options:**

Option Function

**ATT**
Displays the user-definable file attributes.

**DATE**
Displays date and time stamps of files.

**DIR**
Displays only files that have the DIR attribute.

**DRIVE=ALL**
Displays files on all on-line drives.

**DRIVE=(A,B,C,...,P)**
Displays files on the drives specified.

# DIR (continued)

| Option | Function |
|---|---|
| **DRIVE=d** | Displays files on the drive specified by d. |
| **EXCLUDE** | Displays files that DO NOT MATCH the files specified in the command line. |
| **FF** | Sends an initial form feed to the printer device if the printer has been activated by CTRL-P. |
| **FULL** | Shows the name, size, number of 128-byte records, and attributes of the files. If there is a directory label on the drive, DIR shows the password protection mode and the time stamps. If there is no directory label, DIR displays two file entries on a line, omitting the password and time stamp columns. The display is alphabetically sorted. (See SET for a description of file attributes, directory labels, passwords and protection modes.) |
| **LENGTH=n** | Displays n lines of printer output before inserting a table heading. n is a number between 5 and 65536. |
| **MESSAGE** | Displays the names of drives and user numbers DIR is searching. |
| **NOPAGE** | Continuously scrolls information by on the screen. |
| **NOSORT** | Displays files in the order it finds them on the disk. |
| **RO** | Displays only the files that have the Read/Only attribute. |
| **RW** | Displays only the files that are set to Read/Write. |
| **SIZE** | Displays the filename and size in kilobytes (1024 bytes). |

## DIR (continued)

Option Function

**SYS**
Displays only the files that have the SYS attribute.

**USER=ALL**
Displays all files in all user numbers for the default or specified drive.

**USER=n**
Displays the files in the user number specified by n.

**USER=(0,1,...,15)**
Displays files under the user numbers specified.

**Examples:**

A>**DIR C: [FULL]**

Displays full set of characteristics for all files in user 0 on drive C.

A>**DIR C: [DATE]**

Lists the files on drive C and their dates.

A>**DIR D: [RW,SYS]**

Displays all files in user 0 on drive D with Read/Write and System attributes.

3A>**DIR [USER=ALL, DRIVE=ALL]**

Displays all the files in all user numbers (0-15) in all on-line drives.

B6>**DIR [EXCLUDE] *.DAT**

Lists all the files on drive B in user 6 that do not have a filetype of DAT.

3B>**DIR [SIZE] *.PLI *.COM *.ASM**

Displays all the files of type PLI, COM, and ASM in user 3 on drive B in size display format.

## DIR (continued)

**A>DIR [DRIVE=ALL USER=ALL] TESTFILE.BOB**

DIR displays the filename TESTFILE.BOB if it is found on any drive in any user number.

**A>DIR [SIZE,RW] D:**

DIR lists each Read/Write file that resides on Drive D, with its size in kilobytes. Note that D: is equivalent to D:*.*.

# DUMP

**Syntax:**

DUMP filespec

**Purpose:**

DUMP displays the contents of a file in hexadecimal and ASCII format.

**Example:**

A>**DUMP ABC.TEX**

# ED

**Syntax:**

ED
ED input-filespec
ED input-filespec {d:|output-filespec}

**Purpose:**

Character file editor. To redirect or rename the new version of the file specify the destination drive or destination filespec.

**ED Command Summary:**

Command — Action

**nA**
append n lines from original file to memory buffer

**0A**
append file until buffer is one half full

**#A**
append file until buffer is full (or end of file)

**B, -B**
move CP to the beginning (B) or bottom (-B) of buffer

**nC, -nC**
move CP n characters forward (C) or back (-C) through buffer

**nD, -nD**
delete n characters before (-D) or from (D) the CP

**E**
save new file and return to CP/M Plus

**Fstring^Z**
find character string

**H**
save new file, reedit, use new file as original file

**I<RET>**
enter insert mode

**Istring^Z**
insert string at CP

## ED (continued)

Command Action

**Jsearch_str^Zins_str^Zdel_to_str**
juxtapose strings

**nK, -nK**
delete (kill) n lines from the CP

**nL, -nL, 0L**
move CP n lines

**nMcommands**
execute commands n times

**n, -n**
move CP n lines and display that line

**n:**
move to line n

**:ncommand**
execute command through line n

**Nstring^Z**
extended find string

**O**
return to original file

**nP, -nP**
move CP 23 lines forward and display 23 lines at console

**Q**
abandon new file, return to CP/M Plus

**R^Z**
read X$$$$$$$.LIB file into buffer

**Rfilespec^Z**
read filespec into buffer

**Sdelete-string^Zinsert-string**
substitute string

**nT, -nT, 0T**
type n lines

**U, -U**
uppercase translation

**V, -V**
line numbering on/off

**0V**
display free buffer space

## ED (continued)

Command Action

**nW**
write n lines to new file

**0W**
write until buffer is half empty

**nX**
write or append n lines to X$$$$$$$.LIB

**nXfilespec^Z**
write n lines to filespec; append if previous xcommand applied to same file

**0x^Z**
delete file X$$$$$$$.LIB

**0xfilespec^Z**
delete filespec

**nZ**
wait n seconds

**Note:** CP points to the current character being referenced in the edit buffer. Use CTRL-Z to separate multiple commands on the same line. Your screen displays ^Z.

# ERASE

---

**Syntax:**

```
ERASE
ERASE filespec
ERASE filespec [CONFIRM]
```

**Purpose:**

The ERASE command removes one or more files from the directory of a disk. Wildcard characters are accepted in the filespec. Directory and data space are automatically reclaimed for later use by another file. The ERASE command can be abbreviated to ERA.

[CONFIRM] option informs the system to prompt for verification before erasing each file that matches the filespec. CONFIRM can be abbreviated to C.

**Examples:**

A>**ERASE X.PAS**

Removes the file X.PAS from the disk in drive A.

A>**ERA *.PRN**

ERASE *.PRN (Y/N) ?**Y**

All files with the filetype PRN are removed from the disk in drive A.

B>**ERA A:MY*.* [CONFIRM]**

Each file on drive A with a filename that begins with MY is displayed with a question mark for confirmation. Type Y to erase the file displayed, N to keep the file.

A>**ERA B:*.***

ERASE B:*.* (Y/N) ?**Y**

All files on drive B are removed from the disk.

# GENCOM

**Syntax:**

```
GENCOM COM-filespec RSX-filespec...RSX-filespec
         {[LOADER | SCB=(offset,value)]}
GENCOM RSX-filespec ... RSX-filespec
         {[NULL | SCB=(offset,value)]}
GENCOM filename
GENCOM filename [SCB=(offset,value)]
```

**Purpose:**

The GENCOM command attaches RSX files to a COM file, or creates a dummy COM file containing only RSXs. It can also restore a previously GENCOMed file to the original COM file without the header and RSXs, add or replace RSXs in already GENCOMed files, and attach header records to COM files without RSXs.

**GENCOM Options:**

Option Function

**LOADER**
Sets a flag to keep the program loader active.

**NULL**
Indicates that only RSX files are specified. GENCOM creates a dummy COM file for the RSX files. The output COM filename is taken from the filename of the first RSX-filespec.

**SCB=(offset,value)**
Sets the System Control Block from the program by using the hex values specified by (offset,value).

**Examples:**

A>**GENCOM MYPROG PROG1 PROG2**

Generates a new COM file MYPROG.COM with attached RSX's PROG1 and PROG2.

## GENCOM (continued)

A>**GENCOM PROG1 PROG2 [NULL]**

Creates a COM file PROG1.COM with RSX's PROG1 and PROG2.

A>**GENCOM MYPROG**

GENCOM takes MYPROG.COM, strips off the header and deletes all attached RSX's to restore it to its original COM format.

A>**GENCOM MYPROG PROG1 PROG2**

GENCOM looks at the already-GENCOMed file MYPROG.COM to see if PROG1.RSX and PROG2.RSX are already attached RSX files in the module. If either one is already attached, GENCOM replaces it with the new RSX module. Otherwise, GENCOM appends the specified RSX files to the COM file.

# GET

**Syntax:**

GET {CONSOLE INPUT FROM} FILE filespec {options}

GET {CONSOLE INPUT FROM} CONSOLE

**Purpose:**

GET directs the system to take console input from a file for the next system command or user program entered at the console.

Console input is taken from a file until the program terminates. If the file is exhausted before program input is terminated, the program looks for subsequent input from the console. If the program terminates before exhausting all its input, the system reverts back to the console for console input.

**GET Options:**

| Option | Function |
|---|---|
| **ECHO** | Specifies that input is echoed to the console. This is the default option. |
| **NO ECHO** | Specifies that file input is not echoed to the console. The program output and the system prompts are not affected by this option and are still echoed to the console. |
| **SYSTEM** | Specifies that the system immediately go to the specified file for console input. The system reverts to the console for input when it reaches the end of file. You can redirect the system to the console for console input with the GET CONSOLE INPUT FROM CONSOLE command as a command line in the input file. |

## GET (continued)

**Examples:**

A>**GET FILE XINPUT**

A>**MYPROG**

Tells the system to activate the GET utility. Because SYSTEM is not specified, the system reads the next input line from the console and executes MYPROG. If MYPROG program requires console input, it is taken from the file XINPUT. When MYPROG terminates, the system reverts back to the console for console input.

A>**GET FILE XIN2 [SYSTEM]**

Immediately directs the system to get subsequent console input from file XIN2 because it includes the SYSTEM option. The system reverts back to the console for console input when it reaches the end of file in XIN2. Or XIN2 can redirect the system back to the console if it contains a GET CONSOLE command.

A>**GET CONSOLE**

Tells the system to get console input from the console. This command can be used in a file (previously specified in a GET FILE command), which is already being read by the system for console input. It is used to redirect the console input back to the console before the end of file is reached.

# HELP

---

**Syntax:**

```
HELP
HELP topic
HELP topic subtopic
HELP topic [NOPAGE]
HELP topic subtopic1...subtopic8

HELP>topic
HELP>.subtopic
```

**Purpose:**

HELP displays a list of topics and provides summarized information for CP/M Plus commands.

Typing HELP topic displays information about that topic. Typing HELP topic subtopic displays information about that subtopic.

One or two letters is enough to identify the topics. After HELP displays information for your topic, it displays the special prompt HELP> on your screen, followed by a list of subtopics.

- Enter ? to display list of main topics.
- Enter a period and subtopic name to access subtopics.
- Enter a period to redisplay what you just read.
- Press RETURN to return to the CP/M Plus system prompt.
- [NOPAGE] option disables the 24 lines per page console display.
- Press any key to exit a display and return to the HELP> prompt.

**Examples:**

A>**HELP**
A>**HELP DATE**
A>**HELP DIR OPTIONS**
A>**HELP>.OPTIONS**
HELP>**SET**
HELP>**SET PASSWORD**
HELP>**.PASSWORD**
HELP>.
HELP><RET>

# HEXCOM

**Syntax:**

```
HEXCOM filename
```

**Purpose:**

The HEXCOM Command generates a command file (filetype COM) from a HEX input file. It names the output file with the same filename as the input file but with filetype COM. HEXCOM always looks for a file with filetype HEX.

**Example:**

A>**HEXCOM B:PROGRAM**

Generates a command file PROGRAM.COM from the input hex file PROGRAM.HEX on drive B:.

# INITDIR

**Syntax:**

INITDIR d:

**Purpose:**

The INITDIR command initializes a disk directory to allow date and time stamping of files on that disk. INITDIR can also recover time/date directory space.

**Example:**

A>**INITDIR C:**

INITDIR WILL ACTIVATE TIME STAMPS FOR SPECIFIED DRIVE. Do you want to re-format the directory C: (Y/N)?**Y**

# LIB

**Syntax:**

```
LIB filespec {options}
LIB filespec {options}=filespec <modifier>
        {,filespec<modifier> ... }
```

**Purpose:**

A library is a file that contains a collection of object modules. Use the LIB utility to create libraries, and to append, replace, select, or delete modules from an existing library. Use LIB to obtain information about the contents of library files.

LIB creates and maintains library files that contain object modules in Microsoft® REL file format. These modules are produced by the Digital Research® relocatable macro-assembler program, RMAC™, or any other language translator that produces modules in Microsoft REL file format.

You can use LINK-80™ to link the object modules contained in a library to other object files. LINK-80 automatically selects from the library only those modules needed by the program being linked, and then forms an executable file with a filetype of COM.

**LIB Options:**

| Option | Function |
|---|---|
| I | The INDEX option creates an indexed library file of type IRL. LINK-80 searches faster on indexed libraries than on nonindexed libraries. |
| **M** | The MODULE option displays module names. |
| P | The PUBLICS option displays module names and the public variables for the new library file. |
| D | The DUMP option displays the contents of object modules in ASCII form. |

# LIB (continued)

Use modifiers in the command line to instruct LIB to delete, replace, or select modules in a library file. Angle brackets enclose the modules to be deleted or replaced. Parentheses enclose the modules to be selected.

**LIB Modifiers:**

| Modifier | Meaning |
|---|---|
| **Delete** | <module=> |
| **Replace** | <module=filename.REL><br>If module name and filename are the same, you can use the following shorthand:<br><br><filename> |
| **Select** | (modFIRST-modLAST,mod1...modN) |

**Examples:**

A>**LIB TEST4[P]**

Displays all modules and publics in TEST4.REL.

A>**LIB TEST5[P]=FILE1,FILE2**

Creates TEST5.REL from FILE1.REL and FILE2.REL and displays all modules and publics in TEST5.REL.

A>**LIB TEST=TEST1(MOD1,MOD4),TEST2(C1-C4, C6)**

Creates a library file TEST.REL from modules in two source files. TEST1.REL contributes MOD1 and MOD4. LIB extracts modules C1, C4, and all the modules located between them, as well as module C6 from TEST2.REL.

A>**LIB FILE2=FILE3<MODA=>**

Creates FILE2.REL from FILE3.REL, omitting MODA, which is a module in FILE3.REL.

A>**LIB FILE6=FILE5<MODA=FILEB.REL>**

Creates FILE6.REL from FILE5.REL; FILEB.REL replaces MODA.

# LIB (continued)

**A>LIB FILE6=FILE5<THISNAME>**

Module THISNAME is in FILE5.REL. When LIB creates FILE6.REL from FILE5.REL, the file THISNAME.REL replaces the similarly named module THISNAME.

**A>LIB FILE1[I]=B:FILE2(PLOTS,FIND,<br>SEARCH-DISPLAY)**

Creates FILE1.IRL on drive A from the selected modules PLOTS, FIND, and modules SEARCH through the module DISPLAY, in FILE2.REL on drive B.

# LINK

**Syntax:**

```
LINK filespec [options]
LINK filespec [options],...filespec [options]
LINK filespec [options]=filespec [options],...
```

**Purpose:**

LINK combines relocatable object modules such as those produced by RMAC and PL/I-80™ into a COM file ready for execution. Relocatable files can contain external references and publics. Relocatable files can reference modules in library files. LINK searches the library files and includes the referenced modules in the output file. See the Programmer's Utilities Guide for the CP/M® Family of Operating Systems for a complete description of LINK-80.

Use LINK option switches to control execution parameters. Link options follow the file specifications and are enclosed within square brackets. Multiple switches are separated by commas.

**LINK-80 Options:**

| Option | Function |
|---|---|
| **A** | Additional memory; reduces buffer space and writes temporary data to disk. |
| **B** | BIOS link in banked CP/M Plus system. Aligns data segment on page boundary. Puts length of code segment in header. Defaults to SPR filetype. |
| **Dhhhh** | Data origin; sets memory origin for common and data area. |
| **Gn** | Go; set start address to label n. |
| **Lhhhh** | Load; change default load address of module to hhhh. Default 0100H. |
| **Mhhhh** | Memory size; Define free memory requirements for MP/M™ modules. |
| **NL** | No listing of symbol table at console. |
| **NR** | No symbol table file. |

## LINK (continued)

| Option | Function |
| --- | --- |
| **OC** | Output COM command file. Default. |
| **OP** | Output PRL page relocatable file for execution under MP/M in relocatable segment. |
| **OR** | Output RSP resident system process file for execution under MP/M. |
| **OS** | Output SPR system page relocatable file for execution under MP/M. |
| **Phhhh** | Program origin; changes default program origin address to hhhh. Default is 0100H. |
| **Q** | Lists symbols with leading question mark. |
| **S** | Search preceding file as a library. |
| **$Cd** | Destination of console messages. d can be X (console), Y (printer), or Z (zero output). Default is X. |
| **$Id** | Source of intermediate files; d is disk drive A-P. Default is current drive. |
| **$Ld** | Source of library files; d is disk drive A-P. Default is current drive. |
| **$Od** | Destination of object file; d can be Z or disk drive A-P. Default is to same drive as first file in the LINK-80 command. |
| **$Sd** | Destination of symbol file; d can be Y or Z or disk drive A-P. Default is to same drive as first file in LINK-80 command. |

# LINK (continued)

**Examples:**

**A>LINK b:MYFILE[NR]**

LINK-80 on drive A uses as input MYFILE.REL on drive B and produces the executable machine code file MYFILE.COM on drive B. The [NR] option specifies no symbol table file.

**A>LINK m1,m2,m3**

LINK-80 combines the separately compiled files m1, m2, and m3, resolves their external references, and produces the executable machine code file m1.COM.

**A>LINK m=m1,m2,m3**

LINK-80 combines the separately compiled files m1, m2, and m3 and produces the executable machine code file m.COM.

**A>LINK MYFILE,FILE5[s]**

The [s] option tells LINK-80 to search FILE5 as a library. LINK-80 combines MYFILE.REL with the referenced subroutines contained in FILE5.REL on the default drive A and produces MYFILE.COM on drive A.

# MAC

**Syntax:**

MAC filename {$options}

**Purpose:**

MAC™, the CP/M Plus macro assembler, reads assembly language statements from a file of type ASM, assembles the statements, and produces three output files with the input filename and filetypes of HEX, PRN, and SYM. Filename.HEX contains Intel® hexadecimal format object code. Filename.PRN contains an annotated source listing that you can print or examine at the console. Filename.SYM contains a sorted list of symbols defined in the program.

Use options to direct the input and output of MAC. Use a letter with the option to indicate the source and destination drives, and console, printer, or zero output. Valid drive names are A through O. X, P, and Z specify console, printer, and zero output, respectively.

**MAC Options:**

| Option | Function |
|---|---|
| **A** | source drive for ASM file (A-O) |
| **H** | destination drive for HEX file (A-O, Z) |
| **L** | source drive for macrolibrary LIB files called by the MACLIB statement. |
| **P** | destination drive for PRN file (A-O, X, P, Z) |
| **S** | destination drive for SYM file |
| **+L** | lists input lines read from macrolibrary LIB files |
| **-L** | suppresses listing (default) |
| **+M** | lists all macro lines because they are processed during assembly |

# MAC (continued)

**MAC Options:**

| Option | Function |
|---|---|
| **-M** | suppresses all macro lines because they are read during assembly |
| ***M** | lists only hex generated by macro expansions |
| **+Q** | lists all LOCAL symbols in the symbol list |
| **-Q** | suppresses all LOCAL symbols in the symbol list (default) |
| **+S** | appends symbol file to print file |
| **-S** | suppresses creation of symbol file |
| **+1** | produces a pass 1 listing for macro debugging in PRN file |
| **-1** | suppress listing on pass 1 (default) |

**Examples:**

A>**MAC SAMPLE**

MAC assembles the file SAMPLE.ASM.

A>**MAC SAMPLE $PB AA HB SX -M**

In this example, the option list directs the PRN file to drive B:, obtains the ASM file from drive A:, directs the HEX file to drive B:, the SYM file to the console, and suppresses all macro lines during assembly.

# PATCH

**Syntax:**

PATCH filename{.typ} n

**Purpose:**

The PATCH command displays or installs patch number n to the CP/M Plus system or command files. The patch number n must be between 1 and 32 inclusive.

**Example:**

A>**PATCH SHOW 2**

Patches the SHOW.COM system file with patch number 2.

# PIP

**Syntax:**

    Destination = Source

```
PIP
PIP d:{[Gn]}=filespec{options}
PIP filespec{[Gn]}=filespec{options}{, ...}
PIP filespec{[Gn]}|device=filespec{options}|device
```

**Purpose:**

The file copy program PIP copies files, combines files, and transfers files between disks, printers, consoles, or other devices attached to your computer. The first filespec is the destination. The second filespec is the source. Use two or more source filespecs separated by commas to combine two or more files into one file. [o] is any combination of the available options. The [Gn] option in the destination filespec tells PIP to copy your file to that user number. PIP with no command tail displays an * prompt and awaits your series of commands, entered and processed one line at a time. The source or destination can optionally be any CP/M Plus logical device.

**Examples:**

COPY A FILE FROM ONE DISK TO ANOTHER

A>**PIP b:=a:draft.txt**
A>**PIP b:draft.txt = a:**
B3>**PIP myfile.dat=A:[G9]**
A9>**PIP B:[G3]=myfile.dat**

COPY A FILE AND RENAME IT

A5>**PIP newdraft.txt=oldraft.txt**
C8>**PIP b:newdraft.txt=a:oldraft.txt**

COPY MULTIPLE FILES

A>**PIP b:=draft.***
A>**PIP b:=*.***
B>**PIP b:=c:.*.***
C>**PIP a:=*.com[wr]**
B>**PIP a:[g3]=c:*.***

# PIP (continued)

COMBINE MULTIPLE FILES

A>**PIP b:new.dat=file1.dat,file2.dat**

COPY, RENAME AND PLACE IN USER 1

A>**pip newdraft.txt[g1]=oldraft.txt**

COPY, RENAME AND GET FROM USER 1

A>**PIP newdraft.txt=oldraft.txt[g1]**

COPY TO/FROM LOGICAL DEVICES

A>**PIP b:funfile.sue=con:**
A>**PIP lst:=con:**
A>**PIP lst:=b:draft.txt[t8]**
A>**PIP prn:=b:draft.txt**

**PIP Options:**

| Option | Function |
|---|---|
| **A** | Archive. Copy only files that have been changed since the last copy. |
| **C** | Confirm. PIP prompts for confirmation before each file copy. |
| **Dn** | Delete any characters past column n. |
| **E** | Echo transfer to console. |
| **F** | Filter form-feeds from source data. |
| **Gn** | Get from or go to user n. |
| **H** | Test for valid Hex format. |
| **I** | Ignore :00 Hex data records and test for valid Hex format. |
| **L** | Translate uppercase to lowercase. |
| **N** | Number output lines |
| **O** | Object file transfer, ^Z ignored. |
| **Pn** | Set page length to n. (Default n=60). |

# PIP (continued)

**PIP Options:**

| Option | Function |
|---|---|
| **Qs^Z** | Quit copying from source at string s. |
| **R** | Read files that have been set to SYStem. |
| **Ss^Z** | Start copying from the source at the string s. |
| **Tn** | Expand tabs to n spaces. |
| **U** | Translate lowercase to uppercase. |
| **V** | Verify that data has been written correctly. |
| **W** | Write over Read/Only files without console query. |
| **Z** | Zero the parity bit. |

All options except C, Gn, K, O, R, V, and W force an ASCII file transfer, character by character, terminated by a ^Z.

# PUT

**Syntax:**

```
PUT CONSOLE {OUTPUT TO} FILE filespec {option}
PUT PRINTER {OUTPUT TO} FILE filespec {option}
PUT CONSOLE {OUTPUT TO} CONSOLE
PUT PRINTER {OUTPUT TO} PRINTER
```

**Purpose:**

PUT puts console or printer output to a file for the next command entered at the console, until the program terminates. Then console output reverts to the console. Printer output is directed to a file until the program terminates. Then printer output is put back to the printer.

PUT with the SYSTEM option directs all subsequent console/printer output to the specified file. This option terminates when you enter the PUT CONSOLE or PUT PRINTER command.

**PUT Options:**

| Option | Function |
|---|---|
| **ECHO** | Specifies that output is echoed to the console. This is the default option when you direct console output to a file. |
| **NO ECHO** | Specifies that file output is not echoed to the console. NO ECHO is the default for the PUT PRINTER command. |
| **FILTER** | Specifies filtering of control characters, which means that control characters are translated to printable characters. For example, an ESCape character is translated to ^[. |
| **NO FILTER** | Means that PUT does not translate control characters. This is the default option. |

# PUT (continued)

**PUT Options:**

| Option | Function |
|---|---|
| **SYSTEM** | Specifies that system output and program output are written to the file specified by filespec. Output is written to the file until a subsequent PUT CONSOLE command redirects console output back to the console. |

**Examples:**

A>**PUT CONSOLE OUTPUT TO FILE XOUT [ECHO]**

Directs console output to file XOUT with the output echoed to the console.

A>**PUT PRINTER OUTPUT TO FILE XOUT**

A>**MYPROG**

Directs the printer output of program MYPROG to file XOUT. The output is not echoed to the printer.

A>**PUT PRINTER OUTPUT TO FILE XOUT2 [ECHO,SYSTEM]**

Directs all printer output to file XOUT2 as well as to the printer (with ECHO option), and the PUT is in effect until you enter a PUT PRINTER OUTPUT TO PRINTER command.

A>**PUT CONSOLE OUTPUT TO CONSOLE**

Directs console output back to the console.

A>**PUT PRINTER OUTPUT TO PRINTER**

Directs printer output back to the printer.

# RENAME

---

**Syntax:**

```
RENAME
RENAME new-filespec=old-filespec
```

**Purpose:**

RENAME lets you change the name of a file in the directory of a disk. To change several filenames in one command use the * or ? wildcards in the file specifications. You can abbreviate the RENAME command to REN. REN prompts you for input.

**Examples:**

A>**RENAME NEWFILE.BAS=OLDFILE.BAS**

The file OLDFILE.BAS changes to NEWFILE.BAS on drive A.

A>**RENAME**

The system prompts for the following filespecs:

```
Enter New Name:X.PRN
Enter Old Name:Y.PRN
Y       .PRN=X        .PRN
A>
```

File X.PRN is renamed to Y.PRN on drive A.

B>**REN A:PRINTS.NEW=PRINCE.NEW**

The file PRINCE.NEW on drive A changes to PRINTS.NEW on drive A.

A>**RENAME S*.TEX=A*.TEX**

The above command renames all the files matching A*.TEX to files with filenames S*.TEX.

## RENAME (continued)

A>**REN B:NEWLIST=B:OLDLIST**

The file OLDLIST changes to NEWLIST on drive B. Because the second drive specifier, B:, is implied by the first one, it is unnecessary in this example. The command line above has the same effect as the following:

A>**REN B:NEWLIST=OLDLIST**

or

A>**REN NEWLIST=B:OLDLIST**

# RMAC

**Syntax:**

RMAC filespec {options}

**Purpose:**

RMAC, a relocatable macro assembler, assembles ASM files into REL files that you can link to create COM files.

RMAC options specify the destination of the output files. Replace d with the destination drive letter for the output files.

**RMAC Options (d=output option parameter):**

**Rd** drive for REL file (A-O, Z)
**Sd** drive for SYM file (A-O, X, P, Z)
**Pd** drive for PRN file (A-O, X, P, Z)

The d parameter can have the following values:

**A-O** specifies drive A-O
**X** means output to the console
**P** means output to the printer
**Z** means zero output

**Example:**

A>**RMAC TEST $PX SB RB**

Assembles the file TEST.ASM from drive A, sends the listing file (TEST.PRN) to the console, puts the symbol file (TEST.SYM) on drive B and puts the relocatable object file (TEST.REL) on drive B.

# SAVE

**Syntax:**

SAVE

**Purpose:**

SAVE copies the contents of memory to a file. To use SAVE, first issue the SAVE command, then run your program which reads a file into memory. Your program exits to the SAVE utility which prompts you for a filespec to which it copies the contents of memory, and the beginning and ending address of the memory to be SAVEd.

**Example:**

A>**SAVE**

Activates the SAVE utility. Now enter the name of the program which loads a file into memory.

A>**SID dump.com**

Next, execute the program.

**#g0**

When the program exits, SAVE intercepts the return to the system and prompts the user for the filespec and the bounds of memory to be SAVEd.

SAVE Ver 3.0

## SAVE (continued)

Enter file (press RETURN to exit):**dump2.com**

If file DUMP2.COM exists already, the system asks:

Delete dump2.com? **Y**

Then the system asks for the bounds of memory to be saved:

Beginning hex address: **100**

Ending hex address: **400**

The contents of memory from 100H (Hexadecimal) to 400H is copied to file DUMP2.COM.

# SET

**Syntax:**

```
SET [options]
SET d: [options]
SET filespec [options]
SET [option = modifier]
SET filespec [option = modifier]
```

**Purpose:**

SET initiates password protection and time stamping of files. It also sets the file and drive attributes Read/Write, Read/Only, DIR and SYS. It lets you label a disk and password protect the label. To enable time stamping of files, you must first run INITDIR to format the disk directory.

**Examples:**

SET Disk Label operations:

A>**SET [NAME=DISK100]**

Labels the disk on the default drive as DISK100.

A>**SET [PASSWORD=SECRET]**

Assigns SECRET to the disk label.

A>**SET [PASSWORD=<RET>]**

Nullifies the existing password.

**SET Password Operations:**

```
SET [PROTECT=ON]
SET [PROTECT=OFF]
SET filespec [PASSWORD=password]
SET filespec [PROTECT=READ]
SET filespec [PROTECT=WRITE]
SET filespec [PROTECT=DELETE]
SET filespec [PROTECT=NONE]
SET filespec [attribute-options]
```

## SET (continued)

**Password Protection Modifiers:**

| Modifier | Protection |
|---|---|
| **READ** | The password is required for reading, copying writing, deleting or renaming the file. |
| **WRITE** | The password is required for writing, deleting or renaming the file. You do not need a password to read the file. |
| **DELETE** | The password is only required for deleting or renaming the file. You do not need a password to read or modify the file. |
| **NONE** | No password exists for the file. If a password exists, this modifier can be used to delete the password. |

**SET File Attribute Options:**

| Option | Function |
|---|---|
| **RO** | Sets the file attribute to Read/Only. |
| **RW** | Sets the file attribute to Read/Write. |
| **SYS** | Sets the file attribute to SYS. |
| **DIR** | Sets the file attribute to DIR. |
| **ARCHIVE=OFF** | Means that the file has not been backed up (archived). |
| **ARCHIVE=ON** | Means that the file has been backed up (archived). The Archive attribute can be turned on by SET or by PIP when copying a group of files with the PIP [A] option. SHOW and DIR display the Archive option. |
| **F1=ON\|OFF** | Turns on or off the user-definable file attribute F1. |
| **F2=ON\|OFF** | Turns on or off the user-definable file attribute F2. |

# SET (continued)

**SET File Attribute Options:**

| Option | Function |
|---|---|
| **F3=ON\|OFF** | Turns on or off the user-definable file attribute F3. |
| **F4=ON\|OFF** | Turns on or off the user-definable file attribute F4. |

**Examples:**

A>**SET [PROTECT=ON]**

Turns on password protection for all the files on the disk. You must turn on password protection before you can assign passwords to files.

A>**SET [PROTECT=OFF]**

Disables password protection for the files on your disk.

A>**SET MYFILE.TEX [PASSWORD=MYFIL]**

MYFIL is the password assigned to file MYFILE.TEX.

B>**SET *.TEX [PASSWORD=SECRET, PROTECT=WRITE]**

Assigns the password SECRET to all the TEX files on drive B. Each TEX file is given a WRITE protect mode to prevent unauthorized editing.

A>**SET MYFILE.TEX [RO SYS]**

Sets MYFILE.TEX to Read-Only and SYStem.

**SET Default password operation:**

A>**SET [DEFAULT=password]**

Instructs the system to use a default password if you do not enter a password for a password-protected file.

# SET (continued)

**SET Time-stamp Operations:**

**Syntax:**

```
SET{d:} [CREATE=ON|OFF]
SET{d:} [ACCESS=ON|OFF]
SET{d:} [UPDATE=ON|OFF]
```

**Purpose:**

The above SET commands allow you to keep a record of the time and date of file creation and update, or of the last access and update of your files.

**Time and Date Stamp Options:**

| Option | Function |
|---|---|
| **[CREATE=ON]** | Turns on CREATE time stamps on the disk in the default or specified drive. To record the creation time of a file, the CREATE option must be turned on before the file is created. |
| **[ACCESS=ON]** | Turns on ACCESS time stamps on the disk in the default or specified drive. ACCESS and CREATE options are mutually exclusive; only one can be in effect at a time. If you turn on the ACCESS time stamp on a disk that previously had CREATE time stamp, the CREATE time stamp is automatically turned off. |
| **[UPDATE=ON]** | Turns on UPDATE time stamps on the disk in the default or specified drive. UPDATE time stamps record the time the file was last modified. |

# SET (continued)

**Examples:**

```
A>SET [ACCESS=ON]
A>SET [CREATE=ON,UPDATE=ON]
```

**SET Drive Operations:**

**Syntax:**

```
SET {d:} [RO]
SET {d:} [RW]
```

**Example:**

```
A>SET B: [RO]
```

Sets drive B to Read/Only.

# SETDEF

**Syntax:**

```
SETDEF
SETDEF [TEMPORARY=d:]
SETDEF d:{,d:{,d:{,d:}}}
SETDEF [ORDER= (typ {,typ})]
SETDEF [DISPLAY | NO DISPLAY]
SETDEF [PAGE | NOPAGE]
```

**Purpose:**

SETDEF allows the user to display or define up to four drives for the program search order, the drive for temporary files, and the filetype search order. The SETDEF definitions affect only the loading of programs and/or execution of SUBMIT (SUB) files. SETDEF turns on/off the system Display and Console Page modes. When on, the system displays the location and name of programs loaded or SUBmit files executed, and stops after displaying one full console screen of information.

**Examples:**

A>**SETDEF**

Displays current SETDEF parameters.

A>**SETDEF [TEMPORARY=C:]**

Sets disk drive C as the drive to be used for temporary files.

A>**SETDEF C:,***

Tells the system to search for a program on drive C, then, if not found, search for it on the default drive.

A>**SETDEF [ORDER=(SUB,COM)]**

Instructs the system to search for a SUB file to execute. If no SUB file is found, search for a COM file.

## SETDEF (continued)

A>**SETDEF [DISPLAY]**

Turns on the system display mode. The system now displays the name and location of programs loaded or submit files executed.

A>**SETDEF [NO DISPLAY]**

Turns off the system Display mode.

# SHOW

**Syntax:**

```
SHOW
SHOW {d:}
SHOW {d:} [SPACE]
SHOW {d:} [LABEL]
SHOW {d:} [USERS]
SHOW {d:} [DIR]
SHOW {d:} [DRIVE]
```

**Purpose:**

The SHOW command displays the following disk drive information:

- access mode and the amount of free disk space
- disk label
- current user number
- number of files for each user number on the disk
- number of free directory entries for the disk
- drive characteristics

**Examples:**

A>**SHOW**

A>**SHOW [SPACE]**

Instructs the system to display access mode and amount of space left on logged-in drives.

A>**SHOW B:**

Shows access mode for drive B and amount of space left on drive B.

A>**SHOW B:[LABEL]**

Displays label information for drive B.

# SHOW (continued)

**A>SHOW [USERS]**

Displays the current user number and all the users on drive A and the corresponding number of files assigned to them.

**A>SHOW C:[DIR]**

Displays the number of free directory entries on drive C.

**A>SHOW [DRIVE]**

Displays the drive characteristics of drive A.

# SID

**Syntax:**

SID {pgm-filespec}{,sym-filespec}

**Purpose:**

The SID™ symbolic debugger allows you to monitor and test programs developed for the 8080 microprocessor. SID supports real-time breakpoints, fully monitored execution, symbolic disassembly, assembly, and memory display and fill functions. SID can dynamically load SID utility programs to provide traceback and histogram facilities.

**SID Commands:**

Command Meaning

**As**
Enter assembly language statements. s is the start address.

**Cs{b{,d}}**
Call to memory location from SID. s is the called address; b is the value of the BC register pair; d is the value of the DE register pair.

**D{W}{s}{,f}**
Display memory in hex and ASCII. W is a 16-bit word format, s is the start address, and f is the finish address.

**Epgm-filespec {,sym-filespec}**
Load program and symbol table for execution.

**E*sym-filespec**
Load a symbol table file.

**Fs,f,d**
Fill memory with constant value. s is the start address, f is the finish address, and d is an 8-bit data item.

**G{p}{,a{,b}}**
Begin Execution. p is a start address; a is a temporary breakpoint.

# SID (continued)

**SID Commands:**

Command Meaning

**H**
Displays all symbols with addresses in Hex.

**H.a**
Displays hex, decimal, and ASCII values of a where a is a symbolic expression.

**Ha,b**
Computes hex sum and difference of a and b where a and b are symbolic expressions.

**Icommand tail**
Input CCP command line.

**L{s}{,f}**
List 8080 mnemonic instructions. s is the start address, and f is the finish address.

**Ms,h,d**
Move Memory Block. s is the start address, h is the high address of the block, and d is the destination start address.

**P{p{,c}}**
Pass point set, reset, and display. p is a permanent breakpoint address; c is initial value of pass counter.

**Rfilespec{,d}**
Read Code/Symbols. d is an offset to each address.

**S{W}s**
Set Memory Values. s is address where value is sent, and W is 16-bit word.

**T{n{,c}}**
Trace Program Execution. n is the number of program steps, and c is the utility entry address.

**T{W}{n{,c}}**
Trace without Call. W instructs SID not to trace subroutines, n is the number of program steps, and c is the utility entry address.

# SID (continued)

**SID Commands:**

| Command | Meaning |
|---|---|
| **U{W}{n{,c}}** | Monitor Execution without Trace. n is the number of program steps, c is the utility entry address, and W instructs SID not to trace subroutines. |
| **V** | Display the value of the next available location in memory (NEXT), the next location after the largest file read in (MSZE), the current value of the Program counter (PC), and the address of the end-of-available memory (END). |
| **Wfilespec,s,f** | Write the contents of a contiguous block of memory to filespec. f is finish address. |
| **X{f}{r}** | Examine/alter CPU state. f is flag bit C, Z, M, E or I; r is register A, B, D, H, S or P. |

**Examples:**

A>**SID**

CP/M Plus loads SID from drive A into memory. SID displays the # prompt when it is ready to accept commands.

A>**B:SID SAMPLE.HEX**

CP/M Plus loads SID and the program file SAMPLE.HEX into memory from drive B.

**SID Utilities:**

SID utilities, HIST.UTL and TRACE.UTL, are special programs that operate with SID to provide additional debugging facilities. The mechanisms for system initialization, data collection, and data display are described in the <u>Symbolic Instruction Debugger Productivity Tool Reference Manual for the CP/M-80® Family of Operating Systems</u>.

# SID (continued)

The HIST utility creates a histogram (bar graph) showing the relative frequency of execution of code within selected program segments of the test program. The HIST utility allows you to monitor those sections of code that execute most frequently.

The TRACE utility obtains a backtrace of the instructions that led to a particular breakpoint address in a program under test. You can collect the addresses of up to 256 instructions between pass points in U or T modes.

# SUBMIT

**Syntax:**

```
SUBMIT
SUBMIT filespec
SUBMIT filespec argument ... argument
```

**Purpose:**

The SUBMIT command lets you execute a group (batch) of commands from a SUBmit file (a file with filetype of SUB).

**SUB files:**

The SUB file can contain the following types of lines:

- any valid CP/M Plus command
- any valid CP/M Plus command with SUBMIT parameters ($0-$9)
- any data input line
- any program input line with parameters ($0 to $9)

The command line cannot exceed 135 characters.

The following lines illustrate the variety of lines which can be entered in a SUB file:

```
DIR
DIR *.BAK
MAC $1 $$$4
PIP LST:=$1.PRN[T$2 $3 $5]
DIR *.ASM
PIP
<B:=*.ASM
<CON:=DUMP.ASM
<
DIR B:
```

# SUBMIT (continued)

**Examples:**

A>**SUBMIT**

SUBMIT prompts you for the name of the SUB file and any arguments.

A>**SUBMIT SUBA**

SUBMIT executes the commands found in the SUBA.SUB file.

A>**SUBMIT AA ZZ SZ**

SUBMIT executes the commands in AA.SUB, replacing all occurrences of $1 with the argument ZZ and all occurrences of $2 with SZ.

The PROFILE.SUB Start-up File:

Every time you power up or reset your computer, CP/M Plus looks for a special **SUBMIT** file named PROFILE.SUB to execute. If the file does not exist, CP/M Plus resumes normal operation. If the PROFILE.SUB file exists, the system executes the commands in the file. This file is convenient to use if you regularly execute a set of commands before you do your regular session on the computer.

# TYPE

**Syntax:**

```
TYPE
TYPE filespec
TYPE filespec [PAGE]
TYPE filespec [NOPAGE]
```

**Purpose:**

The TYPE command displays the contents of an ASCII character file on your screen.

**TYPE Options:**

| Option | Function |
|---|---|
| **[PAGE]** | Causes the console listing to be displayed in paged mode; that is, stop automatically after listing n lines of text, where n normally defaults to 24 lines per page. |
| **[NOPAGE]** | Turns off Console Page Mode and continuously displays a typed file on the screen. |

**Examples:**

A>**TYPE MYPROG.PLI**

Displays the contents of the file MYPROG.PLI on your screen.

A>**TYPE B:THISFILE [PAGE]**

Displays the contents of the file THISFILE from drive B on your screen 24 lines at a time.

# USER

**Syntax:**

```
USER
USER n
```

**Purpose:**

The USER command sets the current user number. The disk directory can be divided into distinct groups according to a User Number. User numbers range from 0 through 15.

**Examples:**

```
A>USER
Enter User#:5
5A>
```

The current user number is now 5 on drive A.

```
A>USER 3
3A>
```

This command changes the current user number to 3.

# XREF

**Syntax:**

```
XREF {d:} filename {$P}
```

**Purpose:**

XREF™ provides a cross-reference summary of variable usage in a program. XREF requires the PRN and SYM files produced by MAC or RMAC for input to the program. The SYM and PRN files must have the same filename as the filename in the XREF command tail. XREF outputs a file of type XRF.

**Examples:**

A>**XREF b:MYPROG**

XREF operates on the files MYPROG.SYM and MYPROG.PRN on drive B:, producing the file B:MYPROG.XRF.

A>**XREF b:MYPROG $P**

The $P option directs the output to the printer.

# NOTES

# NOTES

## NOTES

DIGITAL RESEARCH®

# SID™

Productivity Tool

# Command Summary

The SID Productivity Tool Command Summary was prepared using the Digital Research TEX™ Text Formatter and printed in the United States of America.

*************************

* First Edition: 1978 *

*************************

# SID™ COMMAND SUMMARY

## STARTUP

(1) SID
(2) SID x.y
(3) SID x.HEX
(4) SID x.UTL
(5) SID x.y u.v

Form (1) starts SID without a test program, (2) loads the test program x.y (y is normally COM), (3) loads x.HEX in Intel "hex" format, (4) loads and executes utility x, (5) loads x.y with the symbol table u.v (normally x.SYM). Example:
SID SORT.COM SORT.SYM

## RESPONSE

(1) #
(2) SYMBOLS
(3) NEXT PC END
nnnn pppp eeee

Form (1) indicates SID is ready to accept commands, (2) indicates machine code loaded, commencing symbol table load, (3) shows successful machine code and/or symbol load where nnnn, pppp, and eeee are hexadecimal values giving the next unfilled machine code location, the initial program counter, and the last free memory location, respectively.

## LETTER COMMANDS

| | | | |
|---|---|---|---|
| A | Assemble | M | Move |
| C | Call | P | Pass Point |
| D | Display | R | Read |
| F | Fill Memory | S | Set Memory |
| G | Go | T | Trace |
| H | Hex | U | Untrace |
| I | Input Line | X | Examine |
| L | List Mnemonics | | |

## COMMAND LINE

SID reads commands from the system console following the # prompt. Each command line is based upon the command letter and optional symbolic expressions. All CP/M®line editing is available on 64 character lines terminated by carriage returns. A space serves as a comma delimiter. SID terminates whenever control-C is typed.

## LITERAL NUMBERS

SID uses the hexadecimal number base, consisting of the decimal digits 0-9 along with the hex digits A-F. Numbers exceeding four digits are truncated to the right. Examples are:

30 3F 3f FF3E F3

## DECIMAL NUMBERS

Decimal numbers are preceded by a #, and consist of decimal digits 0-9. Numbers exceeding 65535 are truncated to the rightmost 16 bits. Examples are:

#48 #9999 #65535 #0

## CHARACTERS

SID accepts graphic ASCII characters within paired string apostrophes ('). Strings of length greater than two are truncated to the right. The rightmost character of a two character string becomes the least significant byte. A one character string has a high order 00 byte, zero length strings are disallowed, and a pair of apostrophes within a string reduces to a single apostrophe. Lower case letters are not translated in strings. Examples are:

'a' 'A' 'xy' '#'''

## SYMBOL REFERENCES

SID symbolic expressions may involve symbol references when a symbol table is present:

(1) .s
(2) @s
(3) =s

Form (1) denotes the address of symbol s, (2) denotes the 16-bit value at .s, (3) denotes the 8-bit value at .s, where s is a sequence of characters matching a symbol table element.

## QUALIFIED SYMBOLS

SID searches for a symbol match starting at the first symbol loaded until the first symbol matches. When duplicate symbols exist, a qualified reference of the form:

$$s_1/s_2/\ .\ .\ .\ /s_n$$

matches symbols from left to right as the search proceeds sequentially through the symbol table. An example is:

ALPHA/GAMMA/I

## SYMBOLIC EXPRESSIONS

Expressions consist of a left to right sequence of literal numbers, decimal numbers, character strings, and symbol references, separated by plus ("+") and minus ("-") operators. Values are added or subtracted, accordingly, with no overflow checks, to produce the final 16-bit result. a leading minus, as in -x, is computed as Ø-x. A leading plus, as in +x, is computed as x'+x, where x' is the value of the last expression typed. A sequence of n ↑'s produces the $n^{th}$ stacked value in the program under test (see the G command). Blanks are not allowed within expressions. Examples are given with individual commands.

## A ASSEMBLE

(1) As
(2) A
(3) -A

Form(1) begins in-line assembly at location s, where each successive address is displayed until a null line or "." is entered by the operator. Form (2) is equivalent to (1) with assumed starting address derived from last assembled, listed, or traced address. Form (3) removes the assembler/disassembler module, discards existing symbol information, and disables subsequent A or L commands. In this case, machine hex code is displayed in subsequent traces. Examples:

A100
A#100
A.CRLF+5
A@GAMMA+@X-=I
A+30

---

## C CALL

(1) Cs
(2) Cs,b
(3) Cs,b,d

Form (1) performs a direct call from SID to location s in memory, without disturbing the CPU state of the program under test, and is most often used with SID Utilities. In this case, registers BC=0000, DE=0000. Form (2) calls s with data BC=b, DE=0000, while form (3) also fills DE=d. Examples:

C100
C#4096
C.DISPLAY
C@JMPVEC+=X
C.CRLF,#34
C.CRLF,@X,+=X

## D DISPLAY MEMORY

(1) Ds
(2) Ds,f
(3) D
(4) D,f
(5) DWs
(6) DWs,f
(7) DW
(8) DW,f

Form (1) types memory contents in 8-bit format starting at location s for ½ screen with graphic ASCII to the right of each line, (2) is similar, but ends at location f. Form (3) continues the display from the last displayed location, or the value of the HL register pair following CPU state display, for ½ screen, (4) is similar, but terminates at location f. Forms (5) through (8) are equivalent to (1) through (4), but display in word format (16-bits). Examples:

```
DF3F
D#100,#200
D.gamma,.DELTA+#30
d,.GAMMA
DW@ALPHA,+#100
```

## F FILL MEMORY

Fs,f,d

Fills memory with 8-bit data d starting at location s, continuing through location f. Examples:

```
F100,3FF,ff
f.gamma,+#100,#23
F@ALPHA,+=I,=X
```

## G GO TO PROGRAM

(1) G
(2) Gp
(3) G,a
(4) Gp,a
(5) G,a,b
(6) Gp,a,b
(7) -G . . .

Form (1) starts the program under test from the current PC without breakpoints. Execution is in real time. Form (2) is equivalent, but sets PC=p before execution, (3) starts from the current PC with a breakpoint at location a, (4) is similar to (3) but sets the PC to p. Form (5) is equivalent to (3) but sets breakpoints at a and b, while (6) presets the PC to p before execution. Upon encountering a breakpoint (or an externally provided RST 7), the break address is printed in the form:

*nnnn

and the optional breakpoints are cleared. Forms given by (7) parallel (1) through (6), except "pass points" are not traced until the corresponding pass count becomes zero (see P command). The symbol "↑" in an expression produces the topmost stacked value, which is used to set a break following a subroutine call. Given that a breakpoint has occurred at a subroutine, the command

G,↑

continues execution with a return breakpoint set. Examples:

```
G1ØØ
G1ØØ,1Ø3
G.CRLF,.PRINT,#1Ø24
G@JMPVEC+=I,.ENDC,.ERRC
G,.errsub
G,.ERRSUB,+3Ø
-G1ØØ,+1Ø,+1Ø
```

## H HEX VALUES

(1) Ha,b
(2) Ha
(3) H

Form (1) produces the hexadecimal sum (a+b) and difference (a-b) of operands. Form (2) performs number conversion by typing the value of a in the format:

hhhh #ddddd 'c' .ssss

where hhhh is a's hex value, dddd is the decimal value, c is the ASCII value, if it exists, and ssss is the symbolic value, if it exists. Form (3) prints the hex values for each symbol table element (abort with rubout). Examples:

H100,200
H#1000,#965
H.GAMMA+=I,@ALPHA-#10
H#53
H@X+=Y-5

## I INPUT LINE

I $c_1c_2 \ldots c_n$

Initializes default low memory areas for the R command or the program under test, as if the characters $c_1$ through $c_n$ had been read and setup at the console command processor level. Default FCB's are initialized, and the default buffer is set to the initial input line.

Examples:

I x.dat
ix.inp y.out
I a:x.inp b:y.out $-p
ITEST.COM
I TEST.HEX TEST.SYM

## L LIST CODE

(1) Ls
(2) Ls,f
(3) L
(4) -L . . .

Form (1) lists disassembled machine code starting at location s for ½ screen, (2) lists mnemonics from location s through f (abort typeouts with rubout). Form (3) lists mnemonics from the last listed, assembled, or traced location for ½ screen. Form (4) parallels (1) through (3), but labels and symbolic operands are not printed. Labels are printed in the form

ssss:

ahead of the lines to which they correspond. Non-8080 mnemonics are printed as

??= hh

where hh is the hex value at that location. Examples:

```
L100
L#1024,#1034
L.CRLF
L@ICALL,+30
-L.PRBUFF+=I,+'A'
```

## M MOVE MEMORY

Ms,h,d

Move data values from start address s through address h to destination address d. Data areas may overlap during the move process. Examples:

```
M100,1FF,300
M.X,.Y,.Z
M.GAMMA,+FF,.DELTA
M@alpha+=x,+#50,+100
```

## P PASS COUNTER

(1) Pp
(2) Pp,c
(3) P
(4) -Pp
(5) -P

A "pass point" is a program counter location to monitor during execution of a test program. A pass point has an associated "pass counter" in the range 1-FF (0-#255) which is decremented each time the test program executes the pass point address. When a pass count reaches 1, the pass point becomes a permanent breakpoint and the pass count remains at 1. Unlike a temporary breakpoint (see G), pass points with pass count 1 stop execution following execution of the instruction at the break address. Form (1) sets a pass point at address p with pass count 1, (2) sets pass point p with pass count c, (3) displays active pass points and counts, (4) clears the pass point at p (equivalent to Pp,0), and (5) clears all pass points. Up to 8 pass points can be active at any time. CPU registers are displayed when executing a pass point, with the header

nn PASS hhhh .ssss

showing the pass count nn and address hhhh with optional symbol ssss. Registers are not displayed if -G or -U is in effect until the pass count reaches 1. Execution can be aborted during the pass trace with rubout. Examples:

```
P100,ff
P.BDOS
P@ICALL+30,#20
-P.CRLF
```

## R READ CODE/SYMBOLS

(1) R
(2) Rd

The I command sets up code and symbol files for subsequent loading with the R command. Form (1) reads optional code and optional symbols in preparation for program test, (2) is similar, but loads code and/or symbols with the bias value d. The sequence:

```
I x.y
R
```

Sets up machine code file x.y (y is usually COM), and reads machine code to the transient area. If y is HEX, the file must be in Intel "hex" format. The sequence:

```
I x.y u.v
R
```

also reads the symbol file u.v (u is usually the same as x, and v is normally SYM). The form:

```
I * u.v
R
```

skips the machine code load, and reads only the symbol file. When a symbol file is specified, the response

SYMBOLS

shows the start of the symbol file read operation. Thus, a "?" error before the SYMBOL message indicates a machine code read error, while "?" following the SYMBOL message shows a symbol file read error. Examples:

```
I COPY.COM
R
I SORT.HEX SORT.SYM
R
I merge.com merge.sym
R1000
I * test.sym
R-#256
```

## S SET MEMORY

(1) Ss
(2) SWs

Form (1) sets memory locations in 8-bit format, (2) sets memory in 16-bit "word" format. In either case, each address is displayed, along with the current content. If a null line is entered, no change is made, and the next address is prompted. If a value is typed, then the data is changed and the next address is prompted. Input terminates with either invalid input, or a single "." from the console. Long ASCII input is entered with form (1) by typing a leading quote (") followed by graphic characters, terminated by a carriage return. The examples show underlined console input:

```
S100
0100 C3 34
0101 24 #254
0102 CF
0103 4B "Ascii
0108 6E =X+5
0109 D4 .
SW.X+#30
2300 006D 44F
2302 4F32 @GAMMA
2304 33E2
2306 FF11 0+.X+=I-#20
2308 348F .
```

## T TRACE MODE

(1) Tn
(2) T
(3) Tn,c
(4) T,c
(5) -T . . .
(6) TW . . .
(7) -TW . . .

Form (1) traces n program steps, showing the CPU state at each step, while (2) traces one step. Form (3) is used with SID utilities, and "calls" the utility function c at each trace step. Form (4) is similar to (3), but traces only one step. Form (5) parallels (1) to (4), but disables symbols. Form (6) parallels (1) to (4), but performs "trace without call" showing only local execution. Form (7) is similar to (6) with symbols disabled. Examples:

```
T100
T#30,.COLLECT
-TW=I,3E03
```

---

## U UNTRACE MODE

(1) U . . .
(2) -U
(3) UW . . .
(4) -UW . . .

U performs the same function as T, except the register state is not displayed. Forms (2) and (4), however, disable intermediate pass point trace (see P). U and T both run fully monitored, with automatic breaks at each instruction. Execution can be aborted with rubout. Examples:

```
Uffff
U#10000,.COLLECT
UW=GAMMA,.COLLECT
```

## X EXAMINE CPU STATE

(1) X
(2) Xf
(3) Xr

Form (1) displays the CPU state in the format:

f A=a B=b D=d H=h S=s P=p i s

where f is the "flag state," a is the 8080 accumulator content, b is the 16-bit BC register pair value, d is the DE value, h is the HL value, s is the SP value, p is the PC value, i is the decoded instruction at p, and s is symbolic information. The flag are represented by dashes ("-") when false, and their letters when true:

Carry Zero Minus Even parity
Interdigit carry

Form (2) allows flag state change, where f is one of C,Z,M,E, or I. The current state is displayed (either "-" or the letter). Enter the value 1 for true, 0 for false, or null for no change. Form (3) allows register state change, where r is one of A, B, D, H, S, or P. Symbol information is given at s when i references an address, including LDAX and STAX. The form "=mm" is printed for memory referencing instructions (e.g., INR M, ADD M), where mm is the memory value before execution. Examples with operator input underlined:

```
XM
M 0
XB
3E04 3EFF
XP
446E .CRLF+10
```

## SID UTILITIES

Utilities execute with SID to provide additional debugging facilities. A utility is loaded initially by typing:

SID x.UTL

where x is the utility name. Upon loading, the utility is setup for execution with SID, and responds with:

.INITIAL = iiii
.COLLECT = cccc
.DISPLAY = dddd

where iiii, cccc, and dddd are three absolute address entries to the utility for (re)initializing, collecting debug data, and displaying collected information, respectively. The SID symbol table contains these three entry names. A utility is reinitialized by typing:

Ciiii or C.INITIAL

The display information is obtained by typing:

Cdddd or C.DISPLAY

while data collection occurs during monitored execution using the T or U commands, where the second argument gives the collection address. Examples are:

Uffff,.collect
U#1000,3403
TW1000,.COLLECT
UW@GAMMA,.COLLECT

Pass points may be set during data collection to stop the monitoring at the end of program areas under test. The actual initialization, collection, and display functions depend upon the particular SID utility.

## THE HIST UTILITY

The HIST utility creates a histogram of program execution between two locations given during initialization. Program addresses are monitored during U or T mode execution, with summary data displayed at any time. Upon startup or reinitialization, HIST prompts with:

TYPE HISTOGRAM BOUNDS:

Respond with:

aaaa,bbbb

for a histogram between locations aaaa and bbbb, inclusive. Collect data in U or T mode, then display results. Output is scaled to the maximum collected value, accumulating until reinitialization. An example:

```
SID HIST.UTL
TYPE HISTOGRAM BOUNDS 100,A00
.INITIAL = 3E03
.COLLECT = 3E06
.DISPLAY = 3E09
#I SORT.COM SORT.SYM
#R
SYMBOLS
#UFF,.COLLECT
(register display and break)
#C.DISPLAY
(histogram display)
U1000,.COLLECT
(display and eventual break)
C.DISPLAY
(updated histogram display)
#C.INITIAL
(histogram bounds reset)
. . .
```

## THE TRACE UTILITY

The TRACE utility provides a dynamic backtrace of up to 256 instructions which ended at the current break address. Instruction address collection occurs only in U or T mode. Pass points can be active, however, during the data collection, and will halt execution when the pass count becomes 1. Initialization clears the accumulated instructions, collection records the instruction address in a wraparound buffer, and display prints the backtrace in decoded mnemonic form with symbol references and labels when they occur. If "-A" is in effect, only instruction addresses are given. In this case, TRACE is loaded by typing:

```
SID
#-A
#I TRACE.UTL
#R
ADDRESSES ONLY
. . .
```

An example of normal operation:

```
SID TRACE.UTL
READY FOR SYMBOLIC BACKTRACE
#I MERGE.COM MERGE.SYM
#R
#UFFF,.COLLECT
(register display, wait, break)
#C.DISPLAY
(symbolic backtrace appears)
. . .
```

## IMPLEMENTATION NOTES

The SID program operates in about 6K bytes, and self-relocates directly below the BDOS (overlaying the CCP area). The SID symbol table fills downward from the base of SID. As the table fills, the BDOS jump address is altered to reflect the reduced free space. Programs which "size" memory using the BDOS jump address should not be started until all symbols are loaded.

The "-A" command increases the free space by about 1½K bytes. Any existing symbol information must be reloaded after issuing the command.

Programs will trace up to the BDOS where tracing is discontinued until control returns to the calling program. ROM subroutine tracing is discontinued when ROM is entered through a call, jump, or PCHL, and resumed upon return to the calling program in RAM.

Use rubout to abort programs running fully monitored in T or U mode, and an externally provided restart (RST 7) when running unmonitored with G.

## 8080 MNEMONICS

The 8080 mnemonics which follow (reproduced with permission from Intel® Corporation), can be entered directly in assembly mode (see A), and are produced by SID in list mode (see L). Data fields can consist of symbolic expressions. Given that "A100" has been typed, and that the symbols X, Y, and Z exist, the following is valid input:

```
MOV  A,B
MVI  A,FF
mvi  b,#255
MVI  M,'x'
LXI  H,'ab'
JMP  100
CALL .X
JZ   @Y
lxi  h,@X+=Z
JMP  .X/Y+5
```

Notable differences between MAC™ and the SID "A" command are that no pseudo operations are allowed, operands are SID symbolic expressions*, labels cannot be inserted, and register references must be names, not numbers.

*In particular, note that
LXI H,'ab'
fills H with 'a' and L with 'b' due to the nature of SID expressions, which is counter to the MAC convention.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| C3 | JMP | | CD | CALL | | C9 | RET |
| C2 | JNZ | | C4 | CNZ | | C0 | RNZ |
| CA | JZ | | CC | CZ | | C8 | RZ |
| D2 | JNC | | D4 | CNC | | D0 | RNC |
| DA | JC | Adr | DC | CC | Adr | D8 | RC |
| E2 | JPO | | E4 | CPO | | E0 | RPO |
| EA | JPE | | EC | CPE | | E8 | RPE |
| F2 | JP | | F4 | CP | | F0 | RP |
| FA | JM | | FC | CM | | F8 | RM |
| E9 | PCHL | | | | | | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 06 | MVI | B, | | C6 | ADI | | 01 | LXI | B, | |
| 0E | MVI | C, | | CE | ACI | | 11 | LXI | D, | D16 |
| 16 | MVI | D, | | D6 | SUI | | 21 | LXI | H, | |
| 1E | MVI | E, | | DE | SBI | | 31 | LXI | SP, | |
| 26 | MVI | H, | D8 | E6 | ANI | D8 | | | | |
| 2E | MVI | L, | | EE | XRI | | | | | |
| 36 | MVI | M, | | F6 | ORI | | | | | |
| 3E | MVI | A, | | FE | CPI | | | | | |

| | | |
|---|---|---|
| 09 | DAD | B |
| 19 | DAD | D |
| 29 | DAD | H |
| 39 | DAD | SP |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 04 | INR | B | 05 | DCR | B | | | |
| 0C | INR | C | 0D | DCR | C | | | |
| 14 | INR | D | 15 | DCR | D | | | |
| 1C | INR | E | 1D | DCR | E | | | |
| 24 | INR | H | 25 | DCR | H | 0A | LDAX | B |
| 2C | INR | L | 2D | DCR | L | 1A | LDAX | D |
| 34 | INR | M | 35 | DCR | M | 2A | LHLD | Adr |
| 3C | INR | A | 3D | DCR | A | 3A | LDA | Adr |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 03 | INX | B | 0B | DCX | B | 02 | STAX | B |
| 13 | INX | D | 1B | DCX | D | 12 | STAX | D |
| 23 | INX | H | 2B | DCX | H | 22 | SHLD | Adr |
| 33 | INX | SP | 3B | DCX | SP | 32 | STA | Adr |

D8 = constant, or logical/arithmetic expression that evaluates to an 8 bit data quantity.

* = all Flags (C, Z, S, P, AC) affected

| | | |
|---|---|---|
| C7 | RST | 0 |
| CF | RST | 1 |
| D7 | RST | 2 |
| DF | RST | 3 |
| E7 | RST | 4 |
| EF | RST | 5 |
| F7 | RST | 6 |
| FF | RST | 7 |
| C5 | PUSH | B |
| D5 | PUSH | D |
| E5 | PUSH | H |
| F5 | PUSH | PSW |
| C1 | POP | B |
| D1 | POP | D |
| E1 | POP | H |
| F1 | POP | PSW* |
| E3 | XTHL | |
| F9 | SPHL | |
| EB | XCHG | |
| 27 | DAA* | |
| 2F | CMA | |
| 37 | STC† | |
| 3F | CMC† | |
| D3 | OUT | D8 |
| DB | IN | D8 |

| | | |
|---|---|---|
| 07 | RLC | |
| 0F | RRC | |
| 17 | RAL | |
| 1F | RAR | |
| 00 | NOP | |
| 76 | HLT | |
| F3 | DI | |
| FB | EI | |
| 40 | MOV | B,B |
| 41 | MOV | B,C |
| 42 | MOV | B,D |
| 43 | MOV | B,E |
| 44 | MOV | B,H |
| 45 | MOV | B,L |
| 46 | MOV | B,M |
| 47 | MOV | B,A |
| 48 | MOV | C,B |
| 49 | MOV | C,C |
| 4A | MOV | C,D |
| 4B | MOV | C,E |
| 4C | MOV | C,H |
| 4D | MOV | C,L |
| 4E | MOV | C,M |
| 4F | MOV | C,A |
| 50 | MOV | D,B |
| 51 | MOV | D,C |
| 52 | MOV | D,D |
| 53 | MOV | D,E |
| 54 | MOV | D,H |
| 55 | MOV | D,L |
| 56 | MOV | D,M |
| 57 | MOV | D,A |

| | | |
|---|---|---|
| 58 | MOV | E,B |
| 59 | MOV | E,C |
| 5A | MOV | E,D |
| 5B | MOV | E,E |
| 5C | MOV | E,H |
| 5D | MOV | E,L |
| 5E | MOV | E,M |
| 5F | MOV | E,A |
| 60 | MOV | H,B |
| 61 | MOV | H,C |
| 62 | MOV | H,D |
| 63 | MOV | H,E |
| 64 | MOV | H,H |
| 65 | MOV | H,L |
| 66 | MOV | H,M |
| 67 | MOV | H,A |
| 68 | MOV | L,B |
| 69 | MOV | L,C |
| 6A | MOV | L,D |
| 6B | MOV | L,E |
| 6C | MOV | L,H |
| 6D | MOV | L,L |
| 6E | MOV | L,M |
| 6F | MOV | L,A |
| 70 | MOV | M,B |
| 71 | MOV | M,C |
| 72 | MOV | M,D |
| 73 | MOV | M,E |
| 74 | MOV | M,H |
| 75 | MOV | M,L |
| | ------------- | |
| 77 | MOV | M,A |
| 78 | MOV | A,B |
| 79 | MOV | A,C |
| 7A | MOV | A,D |
| 7B | MOV | A,E |
| 7C | MOV | A,H |
| 7D | MOV | A,L |
| 7E | MOV | A,M |
| 7F | MOV | A,A |

D16 = constant, or logical/arithmetic expression that evaluates to a 16 bit data quantity.

† = only CARRY affected

| | | |
|---|---|---|
| 80 | ADD | B |
| 81 | ADD | C |
| 82 | ADD | D |
| 83 | ADD | E |
| 84 | ADD | H |
| 85 | ADD | L |
| 86 | ADD | M |
| 87 | ADD | A |
| 88 | ADC | B |
| 89 | ADC | C |
| 8A | ADC | D |
| 8B | ADC | E |
| 8C | ADC | H |
| 8D | ADC | L |
| 8E | ADC | M |
| 8F | ADC | A |
| 90 | SUB | B |
| 91 | SUB | C |
| 92 | SUB | D |
| 93 | SUB | E |
| 94 | SUB | H |
| 95 | SUB | L |
| 96 | SUB | M |
| 97 | SUB | A |
| 98 | SBB | B |
| 99 | SBB | C |
| 9A | SBB | D |
| 9B | SBB | E |
| 9C | SBB | H |
| 9D | SBB | L |
| 9E | SBB | M |
| 9F | SBB | A |
| A0 | ANA | B |
| A1 | ANA | C |
| A2 | ANA | D |
| A3 | ANA | E |
| A4 | ANA | H |
| A5 | ANA | L |
| A6 | ANA | M |
| A7 | ANA | A |
| A8 | XRA | B |
| A9 | XRA | C |
| AA | XRA | D |
| AB | XRA | E |
| AC | XRA | H |
| AD | XRA | L |
| AE | XRA | M |
| AF | XRA | A |
| B0 | ORA | B |
| B1 | ORA | C |
| B2 | ORA | D |
| B3 | ORA | E |
| B4 | ORA | H |
| B5 | ORA | L |
| B6 | ORA | M |
| B7 | ORA | A |
| B8 | CMP | B |
| B9 | CMP | C |
| BA | CMP | D |
| BB | CMP | E |
| BC | CMP | H |
| BD | CMP | L |
| BE | CMP | M |
| BF | CMP | A |

Adr = 16 bit address

** = all Flags except CARRY affected;
(exception: INX & DCX affect no Flags)

# NOTES

# NOTES

# NOTES

# CP/M Plus™

(CP/M® Version 3)

Operating System

# Programmer's Guide

## COPYRIGHT



## DISCLAIMER



## NOTICE TO USER

From time to time changes are made in the filenames and in the files actually included on the distribution disk. This manual should not be construed as a representation or warranty that such files or facilities exist on the distribution disk or as part of the materials and programs distributed. Most distribution disks include a "README.DOC" file. This file explains variations from the manual which do constitute modification of the manual and the items included therewith. Be sure to read this file before using the software.

## TRADEMARKS



The *CP/M Plus (CP/M Version 3) Operating System Programmer's Guide* was printed in the United States of America.

First Edition: January 1983
Second Edition: April 1983

# Foreword

CP/M® 3 is a microcomputer operating system designed for the Intel® 8080, Intel 8085, or other compatible microprocessor. To run CP/M 3, your computer must have an ASCII console, which includes a keyboard and screen, or another display device, from one to sixteen disk drives and a minimum of 32K of memory space. To support additional memory beyond the 64K addressing limit of the processors listed above, CP/M 3 can also support bank-switched memory. The minimum memory requirement for a banked system is 96K.

This manual describes the programming environment of CP/M 3, and is written for experienced programmers who are writing application software in the CP/M 3 environment. It assumes you are familiar with the system features and utilities described in the *CP/M Plus (CP/M Version 3) Operating System User's Guide* and the *Programmer's Utilities Guide for the CP/M Family of Operating Systems.* It also assumes that your CP/M 3 system has been customized for your computer's hardware and is executing as described in the *CP/M Plus (CP/M Version 3) Operating System User's Guide.* If you need to customize your system, please refer to the *CP/M Plus (CP/M Version 3) Operating System System Guide.*

Section 1 of this manual describes the components of the operating system, where they reside in memory, and how they work together to provide a standard operating environment for application programs. Section 2 describes how an application program can call on CP/M 3 to perform serial input and output and manage disk files. Section 3 provides a detailed description of each operating system function. Section 4 presents example programs.

The *CP/M Plus (CP/M Version 3) Operating System Programmer's Guide* contains five appendixes. Appendix A describes the CP/M 3 System Control Block, and defines its fields. Appendix B supplies the format for the Page Relocatable Program. Appendix C tells you how to generate System Page Relocatable files. Appendix D lists the ASCII Symbol Table, and Appendix E summarizes BDOS functions.

# Table of Contents

# Table of Contents (continued)



## Appendixes

# Table of Contents (continued)

## Tables

# Table of Contents (continued)

## Figures

# Section 1
# Introduction to CP/M 3

This section introduces you to the general features of CP/M 3 with an emphasis on how CP/M 3 organizes your computer's memory. The section begins by describing the general memory organization of banked and nonbanked systems and defines the programming environment they have in common. It then shows how CP/M 3 defines memory space into standard regions for operating system modules and executing programs. Subsequent paragraphs describe the components of the operating system, how they communicate with each other and the application program, and in greater detail where each component and program is located in memory. After a brief introduction to disk organization, the final section gives examples of system operation.

CP/M 3 is available in two versions: a version that supports bank-switched memory, and a version that runs on nonbanked systems. CP/M 3 uses the additional memory available in banked systems to provide functions that are not present in the nonbanked version. For example, the banked version of CP/M 3 supports file passwords; the nonbanked version does not. However, because a nonbanked system treats passwords the same way as a banked system does when password protection is not enabled, an application program can run under either system without modification.

## 1.1 Banked and Nonbanked Memory Organization

The memory organization for a nonbanked CP/M 3 system is very simple, as shown in Figure 1-1.

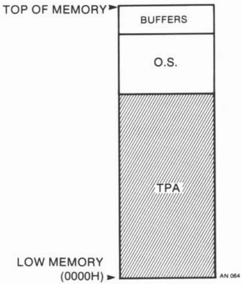


**Figure 1-1. Nonbanked System Memory Organization**

In the nonbanked organization, physical memory consists of a single, contiguous region addressable from 0000H up to a maximum of 0FFFFH (64K-1). The shaded region below the operating system represents the memory space available for the loading and execution of transient programs. The clear area above the operating system represents space that GENCPM can allocate to the operating system for disk record buffers and directory hash tables, as described in the *CP/M Plus (CP/M Version 3) Operating System System Guide*. The minimum size of this area is determined by the specific hardware requirements of the host microcomputer system.

To expand memory capacity beyond the 64K address space of an 8-bit microprocessor, CP/M 3 supports bank-switched memory in a special version called the banked system. In the banked version, the operating system is divided into two modules: the resident portion and the banked portion. The resident portion resides in common memory; the banked portion resides just below the top of banked memory in Bank 0. Figure 1-2 shows memory organization under the banked system.

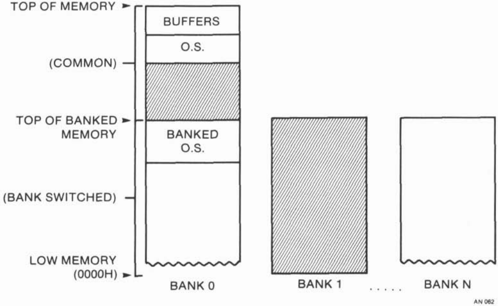


**Figure 1-2. Banked System Memory Organization**

In Figure 1-2, Bank 0 is switched in or in context. The top region of memory, the common region, is always in context; that is, it can always be referenced, no matter what bank is switched in. Figure 1-3 shows memory organization when Bank 1 is in context.

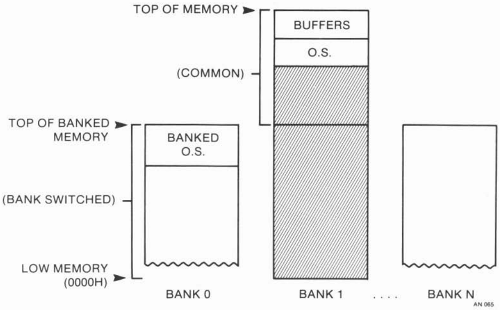


**Figure 1-3. Banked Memory with Bank 1 in Context**

From a transient program's perspective, Bank 1 is always in context. The operating system can switch to Bank 0 or other banks when performing operating system functions without affecting the execution of the transient program. Any bank-switching performed by the operating system is completely transparent to the calling program. Because the major portion of the operating system resides in Bank 0 in banked systems, more memory space is available for transient programs in banked CP/M 3 systems than in nonbanked systems.

The operating system uses the clear areas in Figures 1-2 and 1-3 for disk record buffers and directory hash tables. The clear area in the common region above the operating system represents space that can be allocated for data buffers by GENCPM. Again, the minimum size of this area is determined by the specific hardware requirements of the host microcomputer system.

The banked version of CP/M 3 requires a minimum of two banks, Bank 0 and Bank 1, and can support up to 16 banks of memory. Bank numbers are generally arbitrary with the following exceptions: Bank 0 is the system bank and is in context when CP/M 3 is started. Bank 1 is the transient program bank, and must be contiguous from location zero to the top of banked memory. This requirement does not apply to the other banks. However, common memory must be contiguous.

The size of the common region is typically 16K. The only size requirement on the common region is that it must be large enough to contain the resident portion of the operating system. The maximum top of memory address for both banked and nonbanked systems is 64K-1 (0FFFFH).

In summary, no matter how physical memory is configured, or whether the operating system is banked or nonbanked, CP/M 3 always organizes memory logically so that to a transient program in any CP/M 3 system, memory appears as shown in Figure 1-4.

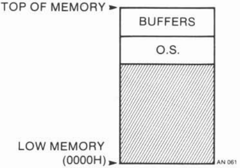


**Figure 1-4. CP/M 3 Logical Memory Organization**

## 1.2 System Components

Functionally, the CP/M 3 operating system is composed of distinct modules. Transient programs can communicate with these modules to request system services. Figure 1-5 shows the regions where these modules reside in logical memory. Note that from the transient program's perspective, Figure 1-5 is just a more detailed version of Figure 1-4.

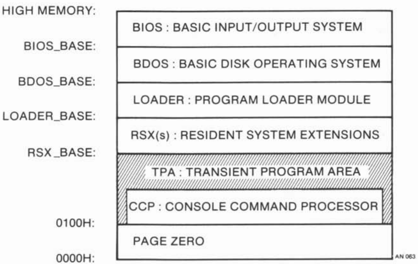


**Figure 1-5. System Components and Regions in Logical Memory**

The Basic Input/Output System, BIOS, is a hardware-dependent module that defines the low-level interface to a particular computer system. It contains the device-driving routines necessary for peripheral device I/O.

The Basic Disk Operating System, BDOS, is the hardware-independent module that is the logical nucleus of CP/M 3. It provides a standard operating environment for transient programs by making services available through numbered system function calls.

The LOADER module handles program loading for the Console Command Processor and transient programs. Usually, this module is not resident when transient programs execute. However, when it is resident, transient programs can access this module by making BDOS Function 59 calls.

Resident System Extensions, RSXs, are temporary additional operating system modules that can selectively extend or modify normal operating system functions. The LOADER module is always resident when RSXs are active.

The Transient Program Area, TPA, is the region of memory where transient programs execute. The CCP also executes in this region.

The Console Command Processor, CCP, is not an operating system module, but is a system program that presents a human-oriented interface to CP/M 3 for the user.

The Page Zero region is not an operating system module either, but functions primarily as an interface to the BDOS module from the CCP and transient programs. It also contains critical system parameters.

## 1.3 System Component Interaction and Communication

This section describes interaction and communication between the modules and regions defined in Section 1.2. The most significant channels of communication are between the BDOS and the BIOS, transient programs and the BDOS, and transient programs and RSXs.

The division of responsibility between the different modules and the way they communicate with one another provide three important benefits. First, because the operating system is divided into two modules—one that is configured for different hardware environments, and one that remains constant on every computer—CP/M 3 software is hardware independent; you can port your programs unchanged to different hardware configurations. Second, because all communication between transient programs and the BDOS is channeled through Page Zero, CP/M 3 transient programs execute, if sufficient memory is available, independent of configured memory size. Third, the CP/M 3 RSX facility can customize the services of CP/M 3 on a selective basis.

### 1.3.1 The BDOS and BIOS

CP/M 3 achieves hardware independence through the interface between the BDOS and the BIOS modules of the operating system. This interface consists of a series of entry points in the BIOS that the BDOS calls to perform hardware-dependent primitive functions such as peripheral device I/O. For example, the BDOS calls the CONIN entry point of the BIOS to read the next console input character.

A system implementor can customize the BIOS to match a specific hardware environment. However, even when the BIOS primitives are customized to match the host computer's hardware environment, the BIOS entry points and the BDOS remain constant. Therefore, the BDOS and the BIOS modules work together to give the CCP and other transient programs hardware-independent access to CP/M 3's facilities.

### 1.3.2 Applications and the BDOS

Transient programs and the CCP access CP/M 3 facilities by making BDOS function calls. BDOS functions can create, delete, open, and close disk files, read or write to opened files, retrieve input from the console, send output to the console or list device, and perform a wide range of other services described in Section 3,"BDOS Functions."

To make a BDOS function call, a transient program loads CPU registers with specific entry parameters and calls location 0005H in Page Zero. If RSXs are not active in memory, location 0005H contains a jump instruction to location BDOS_base + 6. If RSXs are active, location 0005H contains a jump instruction to an address below BDOS_base. Thus, the Page Zero interface allows programs to run without regard to where the operating system modules are located in memory. In addition, transient programs can use the address at location 0006H as a memory ceiling.

Some BDOS functions are similar to BIOS entry points, particularly in the case of simple device I/O. For example, when a transient program makes a console output BDOS function call, the BDOS makes a BIOS console output call. In the case of disk I/O, however, this relationship is more complex. The BDOS might call many BIOS entry points to perform a single BDOS file I/O function.

Transient programs can terminate execution by jumping to location 0000H in the Page Zero region. This location contains a jump instruction to BIOS_base+3, which contains a jump instruction to the BIOS warm start routine. The BIOS warm start routine loads the CCP into memory at location 100H and then passes control to it.

The Console Command Processor is a special system program that executes in the TPA and makes BDOS calls just like an application program. However, the CCP has a unique role: it gives the user access to operating system facilities while transient programs are not executing. It includes several built-in commands, such as TYPE and DIR, that can be executed directly without having to be loaded from disk. When the CCP receives control, it reads the user's command lines, distinguishes between built-in and transient commands, and when necessary, calls upon the LOADER module to load transient programs from disk into the TPA for execution. Section 1.6.2 describes CCP operation in detail.

### 1.3.3 Applications and RSXs

A Resident System Extension is a temporary additional operating system module. An RSX can extend or modify one or more operating system functions selectively. As with a standard BDOS function, a transient program accesses an RSX function through a numbered function call.

At any one time there might be zero, one, or multiple RSXs active in memory. When a transient program makes a BDOS function call, and RSXs are active, each RSX examines the function number of the call. If the function number matches the function the RSX is designed to extend or modify, the RSX performs the requested function. Otherwise, the RSX passes the function request to the next RSX. Nonintercepted functions are eventually passed to the BDOS for standard execution.

RSXs are loaded into memory when programs containing RSXs are loaded. The CP/M 3 utility, GENCOM, can attach RSXs to program files. When attaching RSXs, GENCOM places a special one page header at the beginning of the program file. The CCP reads this header, learns that a program has attached RSXs, and loads the RSXs accordingly. The header itself is not loaded into memory; it merely indicates to the CCP that RSX loading is required.

The LOADER module is a special type of RSX that supports BDOS function 59, Load Overlay. It is always resident when RSXs are active. To indicate RSX support is required, a program that calls function 59 must have an RSX header attached by GENCOM, even if the program does not require other RSXs. When the CCP encounters this type of header in a program file when no RSXs are active, it sets the address at location 0006H in Page Zero to LOADER_base+6 instead of BDOS_base + 6.

## 1.4 Memory Region Boundaries

This section reviews memory regions under CP/M 3, and then describes some details of region boundaries. It then relates the sizes of various modules to the space available for the execution of transient programs. Figure 1-6 reviews the location of regions in logical memory.

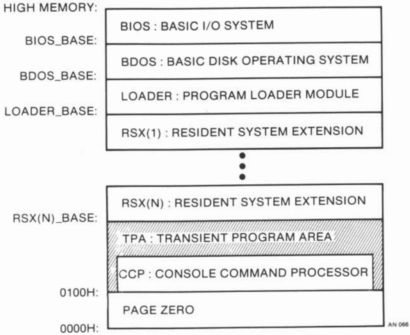


**Figure 1-6. System Modules and Regions in Logical Memory**

First note that all memory regions in CP/M 3 are page-aligned. This means that regions and operating system modules must begin on a page boundary. A page is defined as 256 bytes, so a page boundary always begins at an address where the low-order byte is zero.

The term High Memory in Figure 1-6 denotes the high address of a CP/M 3 system. This address may fall below the actual top of memory address if space above the operating system has been allocated for directory hashing or data buffering by GENCPM. The maximum top of memory address for both banked and nonbanked systems is 64K-1 (0FFFFH).

The labels BIOS_base, BDOS_base, and LOADER_base represent the base addresses of the operating system regions. These addresses always fall on page boundaries. The size of the BIOS region is not fixed, but is determined by the requirements of the host computer system.

The size of the BDOS region differs for the banked and nonbanked versions of CP/M 3. In the banked version, the resident BDOS size is 6 pages, 1.5K. In the nonbanked system, the BDOS size ranges from 31 pages, 7.75K, to 33 pages, 8.25K, depending on system generation options and BIOS requirements.

RSXs are page aligned modules that are stacked in memory below LOADER_base in memory. In the configuration shown in Figure 1-6, location 0005H of Page Zero contains a jump to location RSX(N)_base + 6. Thus, the memory ceiling of the TPA region is reduced when RSXs are active.

Under CP/M 3, the CCP is a transient program that the BIOS loads into the TPA region of memory at system cold and warm start. The BIOS also loads the LOADER module at this time, because the LOADER module is attached to the CCP. When the CCP gains control, it relocates the LOADER module just below BDOS_base. The LOADER module handles program loading for the CCP. It is three pages long.

The maximum size of a transient program that can be loaded into the TPA is limited by LOADER_base because the LOADER cannot load a program over itself. Transient programs may extend beyond this point, however, by using memory above LOADER_base for uninitialized data areas such as I/O buffers. Programs that use memory above BDOS_base cannot make BDOS function calls.

## 1.5 Disk and Drive Organization and Requirements

CP/M 3 can support up to sixteen logical drives, identified by the letters A through P, with up to 512 megabytes of storage each. A logical drive usually corresponds to a physical drive on the system, particularly for physical drives that support removable media such as floppy disks. High-capacity hard disks, however, are commonly divided up into multiple logical drives. Figure 1-7 illustrates the standard organization of a CP/M 3 disk.

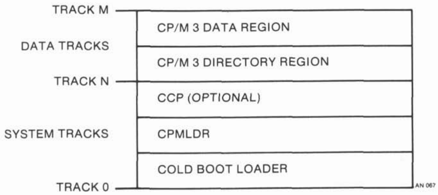


**Figure 1-7. Disk Organization**

In Figure 1-7, the first N tracks are the system tracks. System tracks are required only on the disk used by CP/M 3 during system cold start or warm start. The contents of this region are described in Section 1.6.1. All normal CP/M 3 disk access is directed to the data tracks which CP/M 3 uses for file storage.

The data tracks are divided into two regions: a directory area and a data area. The directory area defines the files that exist on the drive and identifies the data space that belongs to each file. The data area contains the file data defined by the directory. If the drive has adequate storage, a CP/M 3 file can be as large as 32 megabytes.

The directory area is subdivided into sixteen logically independent directories. These directories are identified by user numbers 0 through 15. During system operation, CP/M 3 runs with the user number set to a single value. The user number can be changed at the console with the USER command. A transient program can change the user number by calling a BDOS function.

The user number specifies the currently active directories for all the drives on the system. For example, a PIP command to copy a file from one disk to another gives the destination file the same user number as the source file unless the PIP command is modified by the [G] option.

The directory identifies each file with an eight-character filename and a three-character filetype. Together, these fields must be unique for each file. Files with the same filename and filetype can reside in different user directories on the same drive without conflict. Under the banked version of CP/M 3, a file can be assigned an eight-character password to protect the file from unauthorized access.

All BDOS functions that involve file operations specify the requested file by filename and filetype. Multiple files can be specified by a technique called ambiguous reference, which uses question marks and asterisks as wildcard characters to give CP/M 3 a pattern to match as it searches the directory. A question mark in an ambiguous reference matches any value in the same position in the directory filename or filetype field. An asterisk fills the remainder of the filename or filetype field of the ambiguous reference with question marks. Thus, a filename and filetype field of all question marks, ????????.???, equals an ambiguous reference of two asterisks, *.*, and matches all files in the directory that belong to the current user number.

The CP/M 3 file system automatically allocates directory space and data area space when a file is created or extended, and returns previously allocated space to free space when a file is deleted or truncated. If no directory or data space is available for a requested operation, the BDOS returns an error to the calling program. In general, the allocation and deallocation of disk space is transparent to the calling program. As a result, you need not be concerned with directory and drive organization when using the file system facilities of CP/M 3.

## 1.6 System Operation

This section introduces the general operation of CP/M 3. This overview covers topics concerning the CP/M 3 system components, how they function and how they interact when CP/M 3 is running. This section does not describe the total functionality of CP/M 3, but simply introduces basic CP/M 3 operations.

For the purpose of this overview, CP/M 3 system operation is divided into five categories. First is system cold start, the process that begins execution of the operating system. This procedure ends when the Console Command Processor, CCP, is loaded into memory and the system prompt is displayed on the screen. Second is the operation of the CCP, which provides the user interface to CP/M 3. Third is transient program initiation, execution and termination. Fourth is the way Resident System Extensions run under CP/M 3. The fifth and final category describes the operation of the CP/M 3 SUBMIT utility.

### 1.6.1 Cold Start Operation

The cold start procedure is typically executed immediately after the computer is turned on. The cold start brings CP/M 3 into memory and gives it control of the computer's resources. Cold start is a four-stage procedure.

In the first stage, a hardware feature, or ROM-based software associated with system reset, loads a small program, called the Cold Boot Loader, into memory from the system tracks of drive A (see figure 1-6). The Cold Boot Loader is usually 128 or 256 bytes long.

The Cold Boot Loader performs the second stage of the cold start process. It loads the CP/M 3 loader program, CPMLDR, into memory from the system tracks of the system disk and passes control to it. During this stage, the Cold Boot Loader can also perform other tasks, such as initializing hardware dependent I/O ports.

CPMLDR performs the third stage in the cold start process. First, it reads the CPM3.SYS file from the data area of the disk. The CPM3.SYS file, which is created by the CP/M 3 system generation utility GENCPM, contains the BDOS and BIOS system components and information indicating where these modules are to reside in memory. Once CPMLDR has loaded the BDOS and BIOS into memory, it sends a sign-on message to the console and passes control to the BIOS Cold Boot entry point. If specified as a GENCPM option, CPMLDR can also display a memory map of the CP/M 3 system.

CPMLDR is a small, self-contained version of CP/M 3 that supports only console output and sequential file input. Consistent with CP/M 3's organization, it contains two modules, an invariant CPMLDR_BDOS, and a variant CPMLDR_BIOS that is adapted to match the host microcomputer hardware environment. Cold start initialization of I/O ports and similar functions can also be performed in the CPMLDR_BIOS module during the third stage of cold start.

In the banked version of CP/M 3, these first three stages of the cold boot procedure are performed with Bank 0 in context. The BIOS Cold Start function switches in Bank 1 before proceeding to stage four.

The fourth and final stage in the cold start procedure is performed by the BIOS Cold Start function, Function 0. The entry point to this function is located at BIOS_base as described in Section 1.4. The BIOS Cold Start function begins by performing any remaining hardware initialization, and initializing Page Zero. To initialize Page Zero, the BIOS Cold Start function places a jump to BIOS_base + 3, the BIOS Warm Start entry point, at location 0000H, and a jump to BDOS_base+6, the BDOS entry point, at location 0005H in memory.

The BIOS Cold Start function completes the fourth stage by loading the CCP into the TPA region of memory and passing control to it. The CCP can be loaded from one of two locations. If there is sufficient space in the system tracks for the CCP, it is usually loaded from there. If there is not enough space in the system tracks, the BIOS Cold Start function can read the CCP from the file CCP.COM.

On some banked systems, the CCP is also copied to an alternate bank, so that warm start operations can copy the CCP into the TPA from memory. This speeds up the system warm start operation, and makes it possible to warm start the system without having to access a system disk.

When the CCP gains control, it displays a prompt that references the default disk. If a PROFILE.SUB submit file is present on the default drive, the CCP executes this submit file before prompting the user for a command.

At this point, the cold start procedure is complete. Note that the user number is set to zero when CP/M 3 is cold started. However, the PROFILE submit file can set the user number to another value if this is desirable.

The cold start procedure is designed so that the system tracks need to be initialized only once. This is accomplished because the system track routines are independent of the configured memory size of the CP/M 3 system. The Cold Boot Loader loads CPMLDR into a constant location in memory. This location is chosen when the system is configured. However, CPMLDR locates the BDOS and BIOS system components in memory as specified by the CPM3.SYS file. The CCP always executes at location 100H in the TPA. Thus, CP/M 3 allows the user to generate a new system with GENCPM, and then run it without having to update the system tracks of the system disk.

### 1.6.2 CCP Operation

The Console Command Processor provides the user access to CP/M 3 facilities when transient programs are not running. It also reads the user's command lines, differentiates between built-in commands and transient commands, and executes the commands accordingly.

This section describes the responsibilities and capabilities of the CCP in some detail. The section begins with a description of the CCP's activities when it first receives control from the Cold Start procedure. The section continues with a general discussion of built-in commands, and concludes with a step-by-step description of the procedure the CCP follows to execute the user's commands.

When the CCP gains control following a cold start procedure, it displays the system prompt at the console. This signifies that the CCP is ready to execute a command. The system prompt displays the letter of the drive designated as the initial default drive during GENCPM operation. For example, if drive A was specified as the initial default drive, the CCP displays the following prompt:

```
A>
```

After displaying the system prompt, the CCP scans the directory of the default drive for the file PROFILE.SUB. If the file exists, the CCP creates the command line SUBMIT PROFILE; otherwise the CCP reads the user's first command line by making a BDOS Read Console Buffer function call (BDOS Function 10).

The CCP accepts two different command forms. The simplest CCP command form changes the default drive. The following example illustrates a user changing the default drive from A to B.

```
A>B:
B>
```

This command is one of the CCP's built-in commands. Built-in commands are part of the CCP. They reside in memory while the CCP is active, and therefore can be executed without referencing a disk.

The second command form the CCP accepts is the standard CP/M command line. A standard CP/M command line consists of a command keyword followed by an optional command tail. The command keyword and the command tail can be typed in any combination of upper-case and lower-case letters; the CCP converts all letters in the command line to upper-case. The following syntax defines the standard CP/M command line:

<command> <command tail>

where

| | | |
|---|---|---|
| <command> | => | <filespec> or<br><built-in> |
| <command tail> | => | (no command tail) or<br><filespec> or <filespec><delimiter><filespec> |
| <filespec> | => | {d:}filename{.typ}{;password} |
| <built-in> | => | one of the CCP built-in commands |
| <delimiter> | => | one or more blanks or a tab or one of the following:"=,[]<> " |
| d: | => | CP/M 3 drive specification,"A"through"P" |
| filename | => | 1 to 8 character filename |
| typ | => | 1 to 3 character filetype |
| password | => | 1 to 8 character password value |

Fields enclosed in curly brackets are optional. If there is no drive {d:} present in a file specification <filespec>, the default drive is assumed. If the type field {.typ} is omitted, a type field of all blanks is implied. Omitting the password field {;password} implies a password of all blanks. When a command line is entered at the console, it is terminated by a return or line-feed keystroke.

Transient programs that run under CP/M 3 are not restricted to the above command tail definition. However, the CCP only parses command tails in this format for transient programs. Transient programs that define their command tails differently must perform their own command tail parsing.

The command field must identify either a built-in command, a transient program, or a submit file. For example, USER is the keyword that identifies the built-in command that changes the current user number. The CP/M 3 CCP displays the user number in the system prompt when the user number is non-zero. The following example illustrates changing the user number from zero to 15.

```
B>USER 15
15B>
```

The following table summarizes the built-in commands.

**Table 1-1. CP/M 3 Built-in Commands**

| *Command* | *Meaning* |
|---|---|
| DIR | displays a list of all filenames from a disk directory except those marked with the SYS attribute. |
| DIRSYS | displays a filename list of those files marked with the SYS attribute in the directory. |
| ERASE | erases a filename from a disk directory and releases the storage occupied by the file. |
| RENAME | renames a file. |
| TYPE | displays the contents of an ASCII character file at your console output device. |
| USER | changes from one user number to another. |

Some built-in commands have associated command files which expand upon the options provided by the built-in command. If the CCP reads a command line and discovers the built-in command does not support the options requested in the command line, the CCP loads the built-in function's corresponding command file to perform the command. The DIR command is an example of this type of command. Simple DIR commands are supported by the DIR built-in directly. More complex requests are handled by the DIR.COM utility.

All command keywords that do not identify built-in commands identify either a transient program file or a submit file. If the CCP identifies a command keyword as a transient program, the transient program file is loaded into the TPA from disk and executed. If it recognizes a submit file, the CCP reconstructs the command line into the following form:

SUBMIT <command> <command tail>

and attempts to load and execute the SUBMIT utility. Thus, the original command field becomes the first command tail field of the SUBMIT command. Section 1.6.5 describes the execution of CP/M 3's SUBMIT utility. The procedure the CCP follows to parse a standard command line and execute built-in and transient commands is described as follows:

1. The CCP parses the command line to pick up the command field.
2. If the command field is not preceded by a drive specification, or followed by a filetype or password field, the CCP checks to see if the command is a CCP built-in function. If the command is a built-in command, and the CCP can support the options specified in the command tail, the CCP executes the command. Otherwise, the CCP goes on to step 3.
3. At this point the CCP assumes the command field references a command file or submit file on disk. If the optional filetype field is omitted from the command, the CCP usually assumes the command field references a file of type COM. For example, if the command field is PIP, the CCP attempts to open the file PIP.COM.

   Optionally, the CP/M 3 utility SETDEF can specify that a filetype of SUB also be considered when the command filetype field is omitted. When this automatic submit option is in effect, the CCP attempts to open the command with a filetype of COM. If the COM file cannot be found, the CCP repeats the open operation with a filetype of SUB. As an alternative, the order of open operations can be reversed so that the CCP attempts to open with a filetype of SUB first. In either case, the file that is found on disk first determines the filetype field that is ultimately associated with the command.

   If the filetype field is present in the command, it must equal COM, SUB or PRL. A PRL file is a Page Relocatable file used in Digital Research's multi-user operating system, MP/M™. Under CP/M 3, the CCP handles PRL files exactly like COM files.

If the command field is preceded by a drive specification {d:}, the CCP attempts to open the command or submit file on the specified drive. Otherwise, the CCP attempts to open the file on the drives specified in the drive chain.

The drive chain specifies up to four drives that are to be referenced in sequence for CCP open operations of command and submit files. If an open operation is unsuccessful on a drive in the drive chain because the file cannot be found, the CCP repeats the open operation on the next drive in the chain. This sequence of open operations is repeated until the file is found, or the drive chain is exhausted. The drive chain contains the current default drive as its only drive unless the user modifies the drive chain with the CP/M 3 SETDEF utility.

When the current user number is non-zero, all open requests that fail because the file cannot be found, attempt to locate the command file under user zero. If the file exists under user zero with the system attribute set, the file is opened from user zero. This search for a file under user zero is made by the BDOS Open File function. Thus, the user zero open attempt is made before advancing to the next drive in the search chain.

When automatic submit is in effect, the CCP attempts to open with the first filetype, SUB or COM, on all drives in the search chain before trying the second filetype.

In the banked system, if a password specified in the command field does not match the password of a file on a disk protected in Read mode, the CCP file open operation is terminated with a password error.

If the CCP does not find the command or submit file, it echoes the command line followed by a question mark to the console. If it finds a command file with a filetype of COM or PRL, the CCP proceeds to step 4. If it finds a submit file, it reconstructs the command line as described above, and repeats step 3 for the command, SUBMIT.COM.

4. When the CCP successfully opens the command file, it initializes the following Page Zero fields for access by the loaded transient program:

   0050H : Drive that the command file was loaded from
   0051H : Password address of first file in command tail
   0053H : Password length of first file in command tail
   0054H : Password address of second file in command tail
   0056H : Password length of second file in command tail
   005CH : Parsed FCB for first file in command tail
   006CH : Parsed FCB for second file in command tail
   0080H : Command tail preceded by command tail length

   Page Zero initialization is covered in more detail in Section 2.4.

5. At this point, the CCP calls the LOADER module to load the command file into the TPA. The LOADER module terminates the load operation if a read error occurs, or if the available TPA space is not large enough to contain the file. If no RSXs are resident in memory, the available TPA space is determined by the address LOADER_base because the LOADER cannot load over itself. Otherwise, the maximum TPA address is determined by the base address of the lowest RSX in memory.

6. Once the program is loaded, the LOADER module checks for a RSX header on the program. Programs with RSX headers are identified by a return instruction at location 100H.

   If an RSX header is present, the LOADER relocates all RSXs attached to the end of the program, to the top of the TPA region of memory under the LOADER module, or any other RSXs that are already resident. It also updates the address in location 0006H of Page Zero to address the lowest RSX in memory. Finally, the LOADER discards the RSX header and relocates the program file down one page in memory so that the first executable instruction resides at 100H.

7. After initializing Page Zero, the LOADER module sets up a 32-byte stack with the return address set to location 0000H of Page Zero and jumps to location 100H. At this point, the loaded transient program begins execution.

When a transient program terminates execution, the BIOS warm start routine reloads the CCP into memory. When the CCP receives control, it tests to see if RSXs are resident in memory. If not, it relocates the LOADER module below the BDOS module at the top of the TPA region of memory. Otherwise, it skips this step because the LOADER module is already resident. The CCP execution cycle then repeats.

Unlike earlier versions of CP/M, the CCP does not reset the disk system at warm start. However, the CCP does reset the disk system if a CTRL-C is typed at the prompt.

### 1.6.3 Transient Program Operation

A transient program is one that the CCP loads into the TPA region of memory and executes. As the name transient implies, transient programs are not system resident. The CCP must load a transient program into memory every time the program is to be executed. For example, the utilities PIP and RMAC™ that are shipped with CP/M 3 execute as transient programs; programs such as word processing and accounting packages distributed by applications vendors also execute as transient programs under CP/M 3.

Section 1.6.2 describes how the CCP prepared the CP/M 3 environment for the execution of a transient program. To summarize, the CCP initializes Page Zero to contain parsed command-line fields and sets up a 32-byte stack before jumping to location 0100H to pass control to the transient program. In addition, the CCP might also load RSXs attached to the command file into memory for access by the transient program.

Generally, an executing transient program communicates with the operating system only through BDOS function calls. Transient programs make BDOS function calls by loading the CPU registers with the appropriate entry parameters and calling location 0005H in Page Zero.

Transient programs can use BDOS Function 50, Call BIOS, to access BIOS entry points. This is the preferred method for accessing the BIOS; however, for compatibility with earlier releases of CP/M, transient programs can also make direct BIOS calls for console and list I/O by using the jump instruction at location 0000H in Page Zero. But, to simplify portability, use direct BIOS calls only where the primitive level of functionality provided by the BIOS functions is absolutely required. For example, a disk formatting program must bypass CP/M's disk organization to do its job, and therefore is justified in making direct BIOS calls. Note however, that disk formatting programs are rarely portable.

A transient program can terminate execution in one of three ways: by jumping to location 0000H, by making a BDOS System Reset call, or by making a BDOS Chain To Program call. The first two methods are equivalent; they pass control to the BIOS warm start entry point, which then loads the CCP into the TPA, and the CCP prompts for the next command.

The Chain to Program call allows a transient program to specify the next command to be executed before it terminates its own execution. A Program Chain call executes a standard warm boot sequence, but passes the command specified by the terminating program to the CCP in such a way that the CCP executes the specified command instead of prompting the console for the next command.

Transient programs can also set a Program Return Code before terminating by making a BDOS Function 108 call, Get/Set Program Return Code. The CCP initializes the Program Return Code to zero, successful, when it loads a transient program, unless the program is loaded as the result of a program chain. Therefore, a transient program that terminates successfully can use the Program Return Code to pass a value to a chained program. If the program terminates as the result of a BDOS fatal error, or a CTRL-C entered at the console, the BDOS sets the return code to an unsuccessful value. All other types of program termination leave the return code at its current value.

The CCP has a conditional command facility that uses the Program Return Code. If a command line submitted to the CCP by the SUBMIT utility begins with a colon, the CCP skips execution of the command if the previous command set an unsuccessful Program Return Code. In the following example, the SUBMIT utility sends a command sequence to the CCP:

```
A>SUBMIT SUBFILE
A>COMPUTE RESULTS.DAT
A>:REPORT RESULTS.DAT
```

The CCP does not execute the REPORT command if the COMPUTE command sets an unsuccessful Program Return Code.

### 1.6.4 Resident System Extension Operation

This section gives a general overview of RSX use, then describes how RSXs are loaded, defines the RSX file structure, and tells how the LOADER module uses the RSX prefix and flags to manage RSX activity.

A Resident System Extension (RSX) is a special type of program that can be attached to the operating system to modify or extend the functionality of the BDOS. RSX modules intercept BDOS functions and either perform them, translate them into other BDOS functions, or pass them through untouched. The BDOS executes non-intercepted functions in the standard manner.

A transient program can also use BDOS Function 60, Call Resident System Extension, to call an RSX for special functions. Function 60 is a general purpose function that allows customized interfaces between programs and RSXs.

Two examples of RSX applications are the GET utility and the LOADER module. The GET.COM command file has an attached RSX, GET.RSX, which intercepts all console input calls and returns characters from the file specified in the GET command line. The LOADER module is another example of an RSX, but it is special because it supports Function 59, Load Overlay. It is always resident in memory when other RSXs are active.

RSXs are loaded into memory at program load time. As described in Section 1.6.2, after the CCP locates a command file, it calls the LOADER module to load the program into the TPA. The LOADER loads the transient program into memory along with any attached RSXs. Subsequently, the loader relocates each attached RSX to the top of the TPA and adjusts the TPA size by changing the jump at location 0005H in Page Zero to point to the RSX. When RSX modules reside in memory, the LOADER module resides directly below the BDOS, and the RSX modules stack downward from it.

The order in which the RSX modules are stacked affects the order in which they intercept BDOS calls. A more recently stacked RSX has precedence over an older RSX. Thus, if two RSXs in memory intercept the same BDOS function, the more recently loaded RSX handles the function.

The CP/M 3 utility GENCOM attaches RSX modules to program files. Program files with attached RSXs have a special one page header that the LOADER recognizes when it loads the command file. GENCOM can also attach one or more RSXs to a null command file so that the CCP can load RSXs without having to execute a transient program. In this case, the command file consists of the RSX header followed by the RSXs.

RSX modules are Page Relocatable, PRL, files with the file type RSX. RSX files must be page relocatable because their execution address is determined dynamically by the LOADER module at load time. RSX files have the following format:

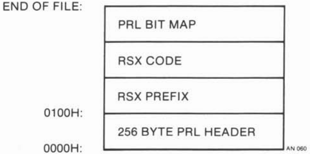


**Figure 1-8. RSX File Format**

RSX files begin with a one page PRL header that specifies the total size of the RSX prefix and code sections. The PRL bit map is a string of bits identifying those bytes in the RSX prefix and code sections that require relocation. The PRL format is described in detail in Appendix B. Note that the PRL header and bit map are removed when an RSX is loaded into memory. They are only used by the LOADER module to load the RSX.

The RSX prefix is a standard data structure that the LOADER module uses to manage RSXs (see Section 4.4). Included in this data structure are jump instructions to the previous and next RSX in memory, and two flags. The LOADER module initializes and updates these jump instructions to maintain the link from location 6 of Page Zero to the BDOS entry point. The RSX flags are the Remove flag and the Nonbanked flag. The Remove flag controls RSX removal from memory. The CCP tests this flag to determine whether or not it should remove the RSX from memory at system warm start. The nonbanked flag identifies RSXs that are loaded only in nonbanked CP/M 3 systems. For example, the CP/M 3 RSX, DIRLBL.RSX, is a nonbanked RSX. It provides BDOS Function 100, Set Directory Label, support for nonbanked systems only. Banked systems support this function in the BDOS.

The RSX code section contains the main body of the RSX. This section always begins with code to intercept the BDOS function that is supported by the RSX. Nonintercepted functions are passed to the next RSX in memory. This section can also include initialization and termination code that transient programs can call with BDOS Function 60.

When the CCP gains control after a system warm start, it removes any RSXs in memory that have the Remove flag set to 0FFH. All other RSXs remain active in memory. Setting an RSX's Remove flag to 0FFH indicates that the RSX is not active and it can be removed. Note that if an RSX marked for removal is not the lowest active RSX in memory, it still occupies memory after removal. Although the removed RSX cannot be executed, its space is returned to the TPA only when all the lower RSXs are removed.

There is one special case where the CCP does not remove an RSX with the Remove flag set to 0FFH following warm start. This case occurs on warm starts following the load of an empty file with attached RSXs. This exception allows an RSX with the Remove flag set to be loaded into memory before a transient program. The transient program can then access the RSX during execution. After the transient program terminates, however, the CCP removes the RSX from the system environment.

As an example of RSX operation, here is a description of the operation of the GET utility. The GET.COM command file has an attached RSX. The LOADER moves this RSX to the top of the TPA when it loads the GET.COM command file. The GET utility performs necessary initializations which include opening the ASCII file specified in the GET command line. It also makes a BDOS Function 60 call to initialize the GET.RSX. At this point, the GET utility terminates. Subsequently, the GET.RSX intercepts all console input calls and returns characters from the file specified in the GET command line. It continues this action until it reads end-of-file. At this point, it sets its Remove flag in the RSX prefix, and stops intercepting console input. On the following warm boot, the CCP removes the RSX from memory.

### 1.6.5 SUBMIT Operation

A SUBMIT command line has the following syntax:

SUBMIT <filespec> <parameters>

If the CCP identifies a command as a submit file, it automatically inserts the SUBMIT keyword into the command line as described in Section 1.6.2.

When the SUBMIT utility begins execution, it opens and reads the file specified by <filespec> and creates a temporary submit file of type $$$ on the system's temporary file drive. GENCPM initializes the temporary file drive to the CCP's current default drive. The SETDEF utility can set the temporary file drive to a specific drive. As it creates the temporary file, SUBMIT performs the parameter substitutions requested by the <parameters> subfield of the SUBMIT command line. See the *CP/M Plus (CP/M Version 3) Operating System User's Guide* for a detailed description of this process.

After SUBMIT creates the temporary submit file, its operation is similar to that of the GET utility described in Section 1.6.4. The SUBMIT command file also has an attached RSX that performs console input redirection from a file. However, the SUBMIT RSX expands upon the simpler facilities provided by the GET RSX. Command lines in a submit file can be marked to indicate whether they are program or CCP input. Furthermore, if a program exhausts all its program input, the next SUBMIT command is a CCP command, the SUBMIT RSX temporarily reverts to console input. Redirected input from the submit file resumes when the program terminates.

Because CP/M 3's submit facility is implemented with RSXs, submit files can be nested. That is, a submit file can contain additional SUBMIT or GET commands. Similarly, a GET command can specify a file that contains GET or SUBMIT commands. For example, when a SUBMIT command is encountered in a submit file, a new SUBMIT RSX is created below the current RSX. The new RSX handles console input until it reads end-of-file on its temporary submit file. At this point, control reverts to the previous SUBMIT RSX.

## 1.7 System Control Block

The System Control Block, SCB, is a 100 byte CP/M 3 data structure that resides in the BDOS system component. The SCB contains internal BDOS flags and data, CCP flags and data, and other system information such as console characteristics and the current date and time. The BDOS, BIOS, CCP system components as well as CP/M 3 utilities and RSXs reference SCB fields. BDOS Function 49, Get/Set System Control Block, provides access to the SCB fields for transient programs, RSXs, and the CCP.

However, use caution when you access the SCB through Function 49 for two reasons. First, the SCB is a CP/M 3 data structure. Digital Research's multi-user operating system, MP/M, does not support BDOS Function 49. Programs that access the SCB can run only on CP/M 3. Secondly, the SCB contains critical system parameters that reflect the current state of the operating system. If a program modifies these parameters illegally, the operating system might crash. However, for application writers who are writing system-oriented applications, access to the SCB variables might prove valuable.

For example, the CCP default drive and current user number are maintained in the System Control Block. This information is displayed in the system prompt. If a transient program changes the current disk or user number by making an explicit BDOS call, the System Control Block values are not changed. They continue to reflect the state of the system when the transient program was loaded. For compatibility with CP/M Version 2, the current disk and user number are also maintained in location 0004H of Page Zero. The high-order nibble contains the user number, and the low-order nibble contains the drive.

Refer to the description of BDOS Function 49 in Section 2.5 for more information on the System Control Block. The SCB fields are also discussed in Appendix A.

*End of Section 1*

# Section 2
# The BDOS System Interface

This section describes the operating system services available to a transient program through the BDOS module of CP/M 3. The section begins by defining how a transient program calls BDOS functions, then discusses serial I/O for console, list and auxiliary devices, the file system, and Page Zero initialization.

## 2.1 BDOS Calling Conventions

CP/M 3 uses a standard convention for BDOS function calls. On entry to the BDOS, register C contains the BDOS function number, and register pair DE contains a byte or word value or an information address. BDOS functions return single-byte values in register A, and double-byte values in register pair HL. In addition, they return with register A equal to L, and register H equal to B. If a transient program makes a BDOS call to a nonsupported function number in the range of 0 to 127, the BDOS returns with register pair HL set to 0FFFFH. For compatibility with MP/M, the BDOS returns with register pair HL set to 0000H on nonsupported function numbers in the range of 128 to 255. Note that CP/M 2 returns with HL set to zero on all invalid function calls. CP/M 3's register passing conventions for BDOS function calls are consistent with the conventions used by the Intel PL/M systems programming language.

When a transient program makes a BDOS function call, the BDOS does not restore registers to their entry values before returning to the calling program. The responsibility for saving and restoring any critical register values rests with the calling program.

When the CCP loads a transient program, the LOADER module sets the stack pointer to a 16 level stack, and then pushes the address 0000H onto the stack. Thus, an immediate return to the system is equivalent to a jump to 0000H. However, most transient programs set up their own stack, and terminate execution by making a BDOS System Reset call (Function 0) or by jumping to location 0000H.

The following example illustrates how a transient program calls a BDOS function. This program reads characters continuously until it encounters an asterisk. Then it terminates execution by returning to the system.

```
bdos    equ     0005h           ;BDOS entry point in Page Zero
conin   equ     1               ;BDOS console input function
;
        org     100h            ;Base of Transient Program Area
nextc:  mvi     c,conin
        call    bdos            ;Return character in A
        cpi     '*'             ;End of processing?
        jnz     nextc           ;Loop if not
        ret                     ;Terminate program
        end
```

## 2.2 BDOS Serial Device I/O

Under CP/M 3, serial device I/O is simply input to and output from simple devices such as consoles, line printers, and communications devices. These physical devices can be assigned the logical device names defined below:

| | |
|---|---|
| CONIN: | logical console input device |
| CONOUT: | logical console output device |
| AUXIN: | logical auxiliary input device |
| AUXOUT: | logical auxiliary output device |
| LST: | logical list output device |

If your system supports the BIOS DEVTBL function, the CP/M 3 DEVICE utility can display and change the assignment of logical devices to physical devices. DEVICE can also display the names and attributes of physical devices supported on your system. If your system does not support the DEVTBL entry point, then the logical to physical device assignments are fixed by the BIOS.

In general, BDOS serial I/O functions read and write an individual ASCII character, or character string to and from these devices, or test the device's ready status. For these BDOS functions, a string of characters is defined as zero to N characters terminated by a delimiter. A block of characters is defined as zero to N characters where N is specified by a word count field. The maximum value of N in both cases is limited only by available memory. The following list summarizes BDOS serial device I/O functions.

Read a character from CONIN:
Read a character buffer from CONIN:
Write a character to CONOUT:
Write a string of characters to CONOUT:
Write a block of characters to CONOUT:
Read a character from AUXIN:
Write a character to AUXOUT:
Write a character to LST:
Write a block of characters to LST:
Interrogate CONIN:, AUXIN:, AUXOUT: ready

CP/M 3 cannot run unless CONIN: and CONOUT: are assigned to a physical console. The remaining logical devices can remain unassigned. If a logical output device is not assigned to a physical device, an output BDOS call to the logical device performs no action. If a logical input device is not assigned to a physical device, an input BDOS call to the logical device typically returns a CTRL-Z (1AH), which indicates end-of-file. Note that these actions depend on your system's BIOS implementation.

### 2.2.1 BDOS Console I/O

Because a transient program's main interaction with its user is through the console, the BDOS supports many console I/O functions. Console I/O functions can be divided into four categories: basic console I/O, direct console I/O, buffered console input, and special console functions. Using the basic console I/O functions, programs can access the console device for simple input and output. The basic console I/O functions are:

1. Console Input - Inputs a single character
2. Console Output- Outputs a single character
9. Print String - Outputs a string of characters
11. Console Status - Signals if a character is ready for input
111. Print Block - Outputs a block of characters

The input function echoes the character to the console so that the user can identify the typed character. The output functions expand tabs in columns of eight characters.

The basic I/O functions also monitor the console to stop and start console output scroll at the user's request. To provide this support, the console output functions make internal status checks for an input character before writing a character to the output device. The console input and console status functions also check the input character. If the user types a CTRL-S, these functions make an additional BIOS console input call. This input call suspends execution until a character is typed. If the typed character is not a CTRL-Q, an additional BIOS console input call is made. Execution and console scrolling resume when the user types a CTRL-Q.

When the BDOS is suspended because of a typed CTRL-S, it scans input for three special characters: CTRL-Q, CTRL-C, and CTRL-P. If the user types any other character, the BDOS echoes a bell character, CTRL-G, to the console, discards the input character, and continues the scan. If the user types a CTRL-C, the BDOS executes a warm start which terminates the calling program. If the user types a CTRL-P, the BDOS toggles the printer echo switch. The printer echo switch controls whether console output is automatically echoed to the list device, LST:. The BDOS signals when it turns on printer echo by sending a bell character to the console.

All basic console I/O functions discard any CTRL-Q or CTRL-P character that is not preceded by a CTRL-S character. Thus, BDOS function 1 cannot read a CTRL-S, CTRL-Q, or CTRL-P character. Furthermore, these characters are invisible to the console status function.

The second category of console I/O is direct console I/O. BDOS function 6 can provide direct console I/O in situations where unadorned console I/O is required. Function 6 actually consists of several sub-functions that support direct console input, output, and status checks. The BDOS does not filter out special characters during direct console I/O. The direct output sub-function does not expand tabs, and the direct input sub-function does not echo typed characters to the console.

The third category of console I/O accepts edited input from the console. The only function in this category, Function 10, Read Buffer Input, reads an input line from a buffer and recognizes certain control characters that edit the input. As an option, the line to be edited can be initialized by the calling program.

In the nonbanked version of CP/M 3, editing within the buffer is restricted to the last character on the line. That is, to edit a character embedded in the line, the user must delete all characters that follow the erroneous character, correct the error, and then retype the remainder of the line. The banked version of CP/M 3 supports complete line editing in which characters can be deleted and inserted anywhere in the line. In addition, the banked version can also recall the previously entered line.

Function 10 also filters input for certain control characters. If the user types a CTRL-C as the first character in the line, Function 10 terminates the calling program by branching to the BIOS warm start entry point. A CTRL-C in any other position is simply echoed at the console. Function 10 also watches for a CTRL-P keystroke, and if it finds one at any position in the command line, it toggles the printer echo switch. Function 10 does not filter CTRL-S and CTRL-Q characters, but accepts them as normal input. In general, all control characters that Function 10 does not recognize as editing control characters, it accepts as input characters. Function 10 identifies a control character with a leading caret, ^, when it echoes the control character to the console. Thus, CTRL-C appears as ^C in a Function 10 command line on the screen.

The final category of console I/O functions includes special functions that modify the behavior of other console functions. These functions are:

109. Get/Set Console Mode
110. Get/Set Output Delimiter

Function 110 can get or set the current delimiter for Function 9, Print String. The delimiter is $, when a transient program begins execution. Function 109 gets or sets a 16-bit system variable called the Console Mode. The following list describes the bits of the Console Mode variable and their functions:

bit 0 : If this bit is set, Function 11 returns true only if a CTRL-C is typed at the console. Programs that make repeated console status calls to test if execution should be interrupted, can set this bit to interrupt on CTRL-C only. The CCP DIR and TYPE built-in commands run in this mode.

bit 1 : Setting this bit disables stop and start scroll support for the basic console I/O functions, which comprise the first category of functions described in this section. When this bit is set, Function 1 reads CTRL-S, CTRL-Q, and CTRL-P, and Function 11 returns true if the user types these characters. Use this mode in situations where raw console input and edited output is needed. While in this mode, you can use Function 6 for input and input status, and Functions 1, 9, and 111 for output without the possibility of the output functions intercepting input CTRL-S, CTRL-Q, or CTRL-P characters.

bit 2 : Setting this bit disables tab expansion and printer echo support for Functions 2, 9, and 111. Use this mode when non-edited output is required.

bit 3 : This bit disables all CTRL-C intercept action in the BDOS. This mode is useful for programs that must control their own termination.

bits 8 and 9 : The BDOS does not use these bits, but reserves them for the CP/M 3 GET RSX that performs console input redirection from a file. With one exception, these bits determine how the GET RSX responds to a program console status request (Function 6, Function 11, or direct BIOS).

bit 8 = 0, bit 9 = 0 - conditional status
bit 8 = 0, bit 9 = 1 - false status
bit 8 = 1, bit 9 = 0 - true status
bit 8 = 1, bit 9 = 1 - do not perform redirection

In conditional status mode, GET responds false to all status requests except for a status call preceded immediately by another status call. On the second call, GET responds with a true result. Thus, a program that spins on status to wait for a character is signaled that a character is ready on the second call. In addition, a program that makes status calls periodically to see if the user wants to stop is not signaled.

When a transient program begins execution, the Console Mode bits are normally set to zero. However, the CP/M 3 utility GENCOM can attach an RSX header to a COM file so that when it is loaded, the console mode bits are set differently. This feature allows you to modify a program's console I/O behavior without having to change the program.

### 2.2.2 Other Serial I/O

The BDOS supports single character output functions for the logical devices LST: and AUXOUT:, an input function for AUXIN:, and status functions for AUXIN: and AUXOUT:. A block output function is also supported for the LST: device. Unlike the console I/O functions, the BDOS does not intercept control characters or expand tabs for these functions. Note that AUXIN: and AUXOUT: replace the READER and PUNCH devices supported by earlier versions of CP/M.

## 2.3 BDOS File System

Transient programs depend on the BDOS file system to create, update, and maintain disk files. This section describes the capabilities of the BDOS file system in detail. You must understand the general features of CP/M 3 described in Section 1 before you can use the detail presented in this section.

The remaining introductory paragraphs define the four categories of BDOS file functions. This is followed by a review of file naming conventions and disk and file organization. The section then describes the data structure used by the BDOS file, and directory oriented functions: the File Control Block (FCB). Subsequent discussions cover file attributes, user numbers, directory labels and extended File Control Blocks (XFCBs), passwords, date and time stamping, blocking and deblocking, multi-sector I/O, disk reset and removable media, byte counts, and error handling. These topics are closely related to the BDOS file system. You must be familiar with the contents of Section 2 before attempting to use the BDOS functions described individually in Section 3.

The BDOS file system supports four categories of functions: file access functions, directory functions, drive related functions, and miscellaneous functions. The file access category includes functions to create a file, open an existing file, and close a file. Both the make and open functions activate the file for subsequent access by BDOS file access functions. The BDOS read and write functions are file access functions that operate either sequentially or randomly by record position. They transfer data in units of 128 bytes, which is the basic record size of the file system. The close function makes any necessary updates to the directory to permanently record the status of an activated file.

BDOS directory functions operate on existing file entries in a drive's directory. This category includes functions to search for one or more files, delete one or more files, truncate a file, rename a file, set file attributes, assign a password to a file, and compute the size of a file. The search and delete functions are the only BDOS functions that support ambiguous file references. All other directory and file related functions require a specific file reference.

The BDOS drive-related category includes functions that select the default drive, compute a drive's free space, interrogate drive status, and assign a directory label to a drive. A drive's directory label controls whether or not CP/M 3 enforces file password protection, or stamps files with the date and time. Note that the nonbanked version of CP/M 3 does not support file passwords.

The miscellaneous category includes functions to set the current DMA address, access and update the current user number, chain to a new program, and flush internal blocking/deblocking buffers. Also included are functions that set the BDOS multi-sector count, and the BDOS error mode. The BDOS multi-sector count determines the number of 128-byte records to be processed by BDOS read and write functions. It can range from 1 to 128. The BDOS error mode determines how the BDOS file system handles certain classes of errors.

Also included in the miscellaneous category are functions that call the BIOS directly, set a program return code, and parse filenames. If the LOADER RSX is resident in memory, programs can also make a BDOS function call to load an overlay. Another miscellaneous function accesses system variables in the System Control Block.

The following list summarizes the operations performed by the BDOS file system:

Disk System Reset
Drive Selection
File Creation
File Open
File Close
Directory Search
File Delete
File Rename
Random or Sequential Read
Random or Sequential Write
Interrogate Selected Disks
Set DMA Address
Set/Reset File Attributes
Reset Drive
Set BDOS Multi-Sector Count
Set BDOS Error Mode
Get Disk Free Space
Chain to Program
Flush Buffers
Get/Set System Control Block
Call BIOS
Load Overlay
Call RSX
Truncate File
Set Directory Label
Get File's Date Stamps and Password Mode
Write File XFCB
Set/Get Date and Time
Set Default Password
Return CP/M 3 Serial Number
Get/Set Program Return Code
Parse Filename

### 2.3.1 File Naming Conventions

Under CP/M 3, a file specification consists of four parts: the drive specifier, the filename field, the filetype field, and the file password field. The general format for a command line file specification is shown below:

{d:}filename{.typ}{;password}

The drive specifier field specifies the drive where the file is located. The filename and type fields identify the file. The password field specifies the password if a file is password protected.

The drive, type, and password fields are optional, and the delimiters :.; are required only when specifying their associated field. The drive specifier can be assigned a letter from A to P where the actual drive letters supported on a given system are determined by the BIOS implementation. When the drive letter is not specified, the current default drive is assumed.

The filename and password fields can contain one to eight non-delimiter characters. The filetype field can contain one to three non-delimiter characters. All three fields are padded with blanks, if necessary. Omitting the optional type or password fields implies a field specification of all blanks.

The CCP calls BDOS Function 152, Parse Filename, to parse file specifications from a command line. Function 152 recognizes certain ASCII characters as valid delimiters when it parses a file from a command line. The valid delimiters are shown in Table 2-1.

**Table 2-1. Valid Filename Delimiters**

| *ASCII* | *HEX EQUIVALENT* |
|---|---|
| null | 00 |
| space | 20 |
| return | 0D |
| tab | 09 |
| : | 3A |
| . | 2E |
| ; | 3B |
| = | 3D |
| , | 2C |
| [ | 5B |
| ] | 5D |
| < | 3C |
| > | 3E |
| \| | 7C |

Function 152 also excludes all control characters from the file fields, and translates all lower-case letters to upper-case.

Avoid using parentheses and the backslash character, \, in the filename and filetype fields because they are commonly used delimiters. Use asterisk and question mark characters, * and ?, only to make an ambiguous file reference. When Function 152 encounters an * in a filename or filetype field, it pads the remainder of the field with question marks. For example, a filename of X*.* is parsed to X???????.???. The BDOS search and delete functions treat a ? in the filename and type fields as follows: A ? in any position matches the corresponding field of any directory entry belonging to the current user number. Thus, a search operation for X???????.??? finds all the current user files on the directory beginning in X. Most other file related BDOS functions treat the presence of a ? in the filename or type field as an error.

It is not mandatory to follow the file naming conventions of CP/M 3 when you create or rename a file with BDOS functions. However, the conventions must be used if the file is to be accessed from a command line. For example, the CCP cannot locate a command file in the directory if its filename or type field contains a lower-case letter.

As a general rule, the filetype field names the generic category of a particular file, while the filename distinguishes individual files in each category. Although they are generally arbitrary, the following list of filetypes names some of the generic categories that have been established.

| | | | |
|---|---|---|---|
| ASM | Assembler Source | PLI | PL/I Source File |
| PRN | Printer Listing | REL | Relocatable Module |
| HEX | Hex Machine Code | TEX | TEX Formatter Source |
| BAS | Basic Source File | BAK | ED Source Backup |
| INT | Intermediate File | SYM | SID Symbol File |
| COM | Command File | $$$ | Temporary File |
| PRL | Page Relocatable | DAT | Data File |
| SPR | Sys. Page Reloc. | SYS | System File |

### 2.3.2 Disk and File Organization

The BDOS file system can support from one to sixteen logical drives. The maximum file size supported on a drive is 32 megabytes. The maximum capacity of a drive is determined by the data block size specified for the drive in the BIOS. The data block size is the basic unit in which the BDOS allocates disk space to files.

Table 2-2 displays the relationship between data block size and drive capacity.

**Table 2-2. Logical Drive Capacity**

| *Data Block Size* | *Maximum Drive Capacity* |
|---|---|
| 1K | 256 Kilobytes |
| 2K | 64 Megabytes |
| 4K | 128 Megabytes |
| 8K | 256 Megabytes |
| 16K | 512 Megabytes |

Logical drives are divided into two regions: a directory area and a data area. The directory area contains from one to sixteen blocks located at the beginning of the drive. The actual number is set in the BIOS. This area contains entries that define which files exist on the drive. The directory entries corresponding to a particular file define those data blocks in the drive's data area that belong to the file. These data blocks contain the file's records. The directory area is logically subdivided into sixteen independent directories identified as user 0 through 15. Each independent directory shares the actual directory area on the drive. However, a file's directory entries cannot exist under more than one user number. In general, only files belonging to the current user number are visible in the directory.

Each disk file consists of a set of up to 262,144 128-byte records. Each record in a file is identified by its position in the file. This position is called the record's random record number. If a file is created sequentially, the first record has a position of zero, while the last record has a position one less than the number of records in the file. Such a file can be read sequentially in record position order beginning at record zero, or randomly by record position. Conversely, if a file is created randomly, records are added to the file by specified position. A file created in this way is called sparse if positions exist within the file where a record has not been written.

The BDOS automatically allocates data blocks to a file to contain its records on the basis of the record positions consumed. Thus, a sparse file that contains two records, one at position zero, the other at position 262,143, consumes only two data blocks in the data area. Sparse files can only be created and accessed randomly, not sequentially. Note that any data block allocated to a file is permanently allocated to the file until the file is deleted or truncated. These are the only mechanisms supported by the BDOS for releasing data blocks belonging to a file.

Source files under CP/M 3 are treated as a sequence of ASCII characters, where each line of the source file is followed by a carriage return line-feed sequence, 0DH followed by 0AH. Thus a single 128-byte record could contain several lines of source text. The end of an ASCII file is denoted by a CTRL-Z character, 1AH, or a real end of file, returned by the BDOS read operation. CTRL-Z characters embedded within machine code files such as COM files are ignored. The actual end-of-file condition returned by the BDOS is used to terminate read operations.

### 2.3.3 File Control Block Definition

The File Control Block, FCB, is a data structure that is set up and initialized by a transient program, and then used by any BDOS file access and directory functions called by the transient program. Thus the FCB is an important channel for information exchange between the BDOS and a transient program. For example, when a program opens a file, and subsequently accesses it with BDOS read and write record functions, the BDOS file system maintains the current file state and position within the program's FCB. Some BDOS functions use certain fields in the FCB for invoking special options. Other BDOS functions use the FCB to return data to the calling program. In addition, all BDOS random I/O functions specify the random record number with a 3-byte field at the end of the FCB.

When a transient program makes a file access or directory BDOS function call, register pair DE must address an FCB. The length of the FCB data area depends on the BDOS function. For most functions, the required length is 33 bytes. For random I/O functions, the Truncate File function, and the Compute File Size function, the FCB length must be 36 bytes. The FCB format is shown on the next page.

| dr | f1 | f2 | ... | f8 | t1 | t2 | t3 | ex | s1 | s2 | rc | d0 | ... | dn | cr | r0 | r1 | r2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | ... | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | ... | 31 | 32 | 33 | 34 | 35 |

where

| | |
|---|---|
| dr | drive code (0 - 16)<br>0 => use default drive for file<br>1 => auto disk select drive A,<br>2 => auto disk select drive B,<br>...<br>16=> auto disk select drive P. |
| f1...f8 | contain the filename in ASCII upper-case, with high bit = 0. f1',..., f8' denote the high-order bit of these positions, and are file attribute bits. |
| t1,t2,t3 | contain the filetype in ASCII upper-case, with high bit = 0. t1', t2', and t3' denote the high bit of these positions, and are file attribute bits.<br>t1' = 1 => Read/Only file<br>t2' = 1 => System file<br>t3' = 1 => File has been archived |
| ex | contains the current extent number, usually set to 0 by the calling program, but can range 0 - 31 during file I/O |
| s1 | reserved for internal system use |
| s2 | reserved for internal system use |
| rc | record count for extent "ex" takes on values from 0 - 255 (values greater than 128 imply record count equals 128) |

| | |
|---|---|
| d0...dn | filled-in by CP/M 3, reserved for system use |
| cr | current record to read or write in a sequential file operation, normally set to zero by the calling program when a file is opened or created |
| r0,r1,r2 | optional random record number in the range 0-262,143 (0 - 3FFFFH). ro,r1,r2 constitute a 18 bit value with low byte r0, middle byte r1, and high byte r2. |

For BDOS directory functions, the calling program must initialize bytes 0 through 11 of the FCB before issuing the function call. The Set Directory Label and Write File XFCB functions also require the calling program to initialize byte 12. The Rename File function requires the calling program to place the new filename and type in bytes 17 through 27.

BDOS open or make function calls require the calling program to intialize bytes 0 through 12 of the FCB before making the call. Usually, byte 12 is set to zero. In addition, if the file is to be processed from the beginning using sequential read or write functions, byte 32, cr, must be zeroed.

After an FCB is activated by an open or make operation, a program does not have to modify the FCB to perform sequential read or write operations. In fact, bytes 0 through 31 of an activated FCB should not be modified. However, random I/O functions require that a program set bytes 33 through 35 to the requested random record number prior to making the function call.

File directory entries maintained in the directory area of each disk have the same format as FCBs, excluding bytes 32 through 35, except for byte 0 which contains the file's user number. Both the Open File and Make File functions bring these entries, excluding byte 0, into memory in the FCB specified by the calling program. All read and write operations on a file must specify an FCB activated in this manner.

The BDOS updates the memory copy of the FCB during file processing to maintain the current position within the file. During file write operations, the BDOS updates the memory copy of the FCB to record the allocation of data to the file, and at the termination of file processing, the Close File function permanently records this information on disk. Note that data allocated to a file during file write operations is not completely recorded in the directory until the calling program issues a Close File call. Therefore, a program that creates or modifies files must close the files at the end of any write processing. Otherwise, data might be lost.

The BDOS Search and Delete functions support multiple or ambiguous file references. In general, a question mark in the filename, filetype, or extent field matches any value in the corresponding positions of directory FCBs during a directory search operation. The BDOS search functions also recognize a question mark in the drive code field, and if specified, they return all directory entries on the disk regardless of user number, including empty entries. A directory FCB that begins with E5H is an empty directory entry.

### 2.3.4 File Attributes

The high-order bits of the FCB filename, f1',...,f8', and filetype, t1',t2',t3', fields are called attribute bits. Attributes bits are 1 bit Boolean fields where 1 indicates on or true, and 0 indicates off or false. Attribute bits indicate two kinds of attributes within the file system: file attributes and interface attributes.

The file attribute bits, f1',...,f4' and t1',t2',t3', can indicate that a file has a defined file attribute. These bits are recorded in a file's directory FCBs. File attributes can be set or reset only by the BDOS Set File Attributes function. When the BDOS Make File function creates a file, it initializes all file attributes to zero. A program can interrogate file attributes in an FCB activated by the BDOS Open File function, or in directory FCBs returned by the BDOS Search For First and Search For Next functions.

Note: the BDOS file system ignores file attribute bits when it attempts to locate a file in the directory.

The file system defines the file attribute bits, t1',t2',t3', as follows:

t1': Read-Only attribute - The file system prevents write operations to a file with the read-only attribute set.

t2': System attribute - This attribute, if set, identifies the file as a CP/M 3 system file. System files are not usually displayed by the CP/M 3 DIR command. In addition, user-zero system files can be accessed on a read-only basis from other user numbers.

t3': Archive attribute - This attribute is designed for user written archive programs. When an archive program copies a file to backup storage, it sets the archive attribute of the copied files. The file system automatically resets the archive attribute of a directory FCB that has been issued a write command. The archive program can test this attribute in each of the file's directory FCBs via the BDOS Search and Search Next functions. If all directory FCBs have the archive attribute set, it indicates that the file has not been modified since the previous archive. Note that the CP/M 3 PIP utility supports file archival.

Attributes f1' through f4' are available for definition by the user.

The interface attributes are indicated by bits f5' through f8' and cannot be used as file attributes. Interface attributes f5' and f6' can request options for BDOS Make File, Close File, Delete File, and Set File Attributes functions. Table 2-3 defines options indicated by the f5' and f6' interface attribute bits for these functions.

**Table 2-3. BDOS Interface Attributes**

| *BDOS Function* | *Interface Attribute Definition* |
|---|---|
| 16. Close File | f5' = 1 : Partial Close |
| 19. Delete File | f5' = 1 : Delete file XFCBs only |
| 22. Make File | f6' = 1 : Assign password to file |
| 30. Set File Attributes | f6' = 1 : Set file byte count |

Section 3 discusses each interface attribute in detail in the definitions of the above functions. Attributes f5' and f6' are always reset when control is returned to the calling program. Interface attributes f7' and f8' are reserved for internal use by the BDOS file system.

### 2.3.5 User Number Conventions

The CP/M 3 User facility divides each drive directory into sixteen logically independent directories, designated as user 0 through user 15. Physically, all user directories share the directory area of a drive. In most other aspects, however, they are independent. For example, files with the same name can exist on different user numbers of the same drive with no conflict. However, a single file cannot reside under more than one user number.

Only one user number is active for a program at one time, and the current user number applies to all drives on the system. Furthermore, the FCB format does not contain any field that can be used to override the current user number. As a result, all file and directory operations reference directories associated with the current user number. However, it is possible for a program to access files on different user numbers; this can be accomplished by setting the user number to the file's user number with the BDOS Set User function before making the desired BDOS function call for the file. Note that this technique must be used carefully. An error occurs if a program attempts to read or write to a file under a user number different from the user number that was active when the file was opened.

When the CCP loads and executes a transient program, it initializes the user number to the value displayed in the system prompt. If the system prompt does not display a user number, user zero is implied. A transient program can change its user number by making a BDOS Set User function call. Changing the user number in this way does not affect the CCP's user number displayed in the system prompt. When the transient program terminates, the CCP's user number is restored. However, an option of the BDOS Program Chain command allows a program to pass its current user number and default drive to the chained program.

User 0 has special properties under CP/M 3. When the current user number is not equal to zero, and if a requested file is not present under the current user number, the file system automatically attempts to open the file under user zero. If the file exists under user zero, and if it has the system attribute, t2', set, the file is opened from user zero. Note, however, that files opened in this way cannot be written to; they are available only for read access. This procedure allows utilities that may include overlays and any other commonly accessed files to be placed on user zero, but also be available for access from other user numbers. As a result, commonly needed utilities need not be copied to all user numbers on a directory, and you can control which user zero files are directly accessible from other user numbers.

### 2.3.6 Directory Labels and XFCBs

The BDOS file system includes two special types of FCBs: the XFCB and the Directory Label. The XFCB is an extended FCB that optionally can be associated with a file in the directory. If present, it contains the file's password. Note that password protected files and XFCBs are supported only in the banked version of CP/M 3. The format of the XFCB follows.

| DR | FILE | TYPE | PM | S1 | S2 | RC | PASSWORD | RESERVED |
|---|---|---|---|---|---|---|---|---|
| 00 | 01 . . . | 09 . . | 12 | 13 | 14 | 15 | 16 . . . . . . | 24 . . . . . . |

AN 069

**Figure 2-1. XFCB Format**

| | | |
|---|---|---|
| dr | - | drive code (0 - 16) |
| file | - | filename field |
| type | - | filetype field |
| pm | - | password mode |
| | | bit 7 - Read mode |
| | | bit 6 - Write mode |
| | | bit 5 - Delete mode |
| | | ** - bit references are right to left, relative to 0 |
| s1,s2,rc | - | reserved for system use |
| password | - | 8-byte password field (encrypted) |
| reserved | - | 8-byte reserved area |

An XFCB can be created only on a drive that has a directory label, and only if the directory label has activated password protection. For drives in this state, an XFCB can be created for a file in two ways: by the BDOS Make function or by the BDOS Write File XFCB function. The BDOS Make function creates an XFCB if the calling program requests that a password be assigned to the created file. The BDOS Write File XFCB function can be used to assign a password to an existing file. Note that in the directory, an XFCB is identified by a drive byte value, byte 0 in the FCB, equal to 16 + N, where N equals the user number.

For its drive, the directory label specifies if file password support is to be activated, and if date and time stamping for files is to be performed. The format of the Directory Label follows.

| DR | NAME | TYPE | D1 | S1 | S2 | RC | PASSWORD | TS1 | TS2 |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 . . | 09 . . | 12 | 13 | 14 | 15 | 16 . . . . . . | 24 . | 28 . |

AN 070

**Figure 2-2. Directory Label Format**

| | | |
|---|---|---|
| dr | - | drive code (0 - 16) |
| name | - | Directory Label name |
| type | - | Directory Label type |
| dl | - | Directory Label data byte<br>bit 7 - require passwords for password protected files<br>bit 6 - perform access time stamping<br>bit 5 - perform update time stamping<br>bit 4 - perform create time stamping<br>bit 0 - Directory Label exists<br>** - bit references are right to left, relative to 0 |
| s1,s2,rc | - | n/a |
| password | - | 8-byte password field (encrypted) |
| ts1 | - | 4-byte creation time stamp field |
| ts2 | - | 4-byte update time stamp field |

Only one Directory Label can exist in a drive's directory. The Directory Label name and type fields are not used to search for a Directory Label; they can be used to identify a disk. A Directory Label can be created, or its fields can be updated by BDOS function 100, Set Directory Label. This function can also assign a Directory Label a password. The Directory Label password, if assigned, cannot be circumvented, whereas file password protection is an option controlled by the Directory Label. Thus, access to the Directory Label password provides a kind of super-user status on that drive.

The nonbanked version of CP/M 3 does not support file passwords. However, it does provide password protection of directory labels. The CP/M 3 RSX, DIRLBL.RSX, which implements BDOS Function 100 in the nonbanked version of CP/M 3, provides this support.

The BDOS file system has no function to read the Directory Label FCB directly. However, the Directory Label data byte can be read directly with the BDOS Function 101, Return Directory Label. In addition, the BDOS Search functions, with a ? in the FCB drive byte, can be used to find the Directory Label on the default drive. In the directory, the Directory Label is identified by a drive byte value, byte 0 in the FCB, equal to 32, 20H.

### 2.3.7 File Passwords

Only the banked version of CP/M 3 supports file passwords. In the nonbanked version, all BDOS functions with password related options operate the same way the banked version does when passwords are not enabled.

Files can be assigned passwords in two ways: by the Make File function or by the Write File XFCB function. A file's password can also be changed by the Write File XFCB function if the original password is supplied.

Password protection is provided in one of three modes. Table 2-4 shows the difference in access level allowed to BDOS functions when the password is not supplied.

**Table 2-4. Password Protection Modes**

| *Password Mode* | *Access level allowed when the password is not supplied.* |
|---|---|
| 1. Read | The file cannot be read. |
| 2. Write | The file can be read, but not modified. |
| 3. Delete | The file can be modified, but not deleted. |

If a file is password protected in Read mode, the password must be supplied to open the file. A file protected in Write mode cannot be written to without the password. A file protected in Delete mode allows read and write access, but the user must specify the password to delete the file, rename the file, or to modify the file's attributes. Thus, password protection in mode 1 implies mode 2 and 3 protection, and mode 2 protection implies mode 3 protection. All three modes require the user to specify the password to delete the file, rename the file, or to modify the file's attributes.

If the correct password is supplied, or if password protection is disabled by the Directory Label, then access to the BDOS functions is the same as for a file that is not password protected. In addition, the Search For First and Search For Next functions are not affected by file passwords.

Table 2-5 lists the BDOS functions that test for password.

**Table 2-5. BDOS Functions That Test For Password**

| | |
|---|---|
| 15. | Open File |
| 19. | Delete File |
| 23. | Rename File |
| 30. | Set File Attributes |
| 99. | Truncate File |
| 100. | Set Directory Label |
| 103. | Write File XFCB |

File passwords are eight bytes in length. They are maintained in the XFCB Directory Label in encrypted form. To make a BDOS function call for a file that requires a password, a program must place the password in the first eight bytes of the current DMA, or specify it with the BDOS function, Set Default Password, prior to making the function call.
**Note:** the BDOS keeps an assigned default password value until it is replaced with a new assigned value.

### 2.3.8 File Date and Time Stamps

The CP/M 3 File System uses a special type of directory entry called an SFCB to record date and time stamps for files. When a directory has been initialized for date and time stamping, SFCBs reside in every fourth position of the directory. Each SFCB maintains the date and time stamps for the previous three directory entries as shown in Figure 2-3.

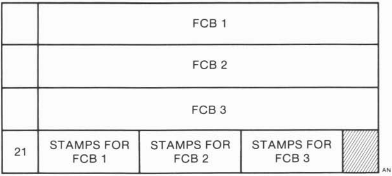


**Figure 2-3. Directory Record with SFCB**

This figure shows a directory record that contains an SFCB. Directory records consist of four directory entries, each 32 bytes long. SFCBs always occupy the last position of a directory record.

The SFCB directory item contains five fields. The first field is one byte long and contains the value 21H. This value identifies the SFCB in the directory. The next three fields, the SFCB subfields, contain the date and time stamps for their corresponding FCB entries in the directory record. These fields are 10 bytes long. The last byte of the SFCB is reserved for system use. The format of the SFCB subfields is shown in Table 2-6.

**Table 2-6. SFCB Subfields Format**

| *Offset in Bytes* | | *SFCB Subfield Contents* |
|---|---|---|
| 0 — 3 | : | Create or Access Date and Time Stamp field |
| 4 — 7 | : | Update Date and Time Stamp field |
| 8 | : | Password mode field |
| 9 | : | Reserved |

An SFCB subfield contains valid information only if its corresponding FCB in the directory record is an extent zero FCB. This FCB is a file's first directory entry. For password protected files, the SFCB subfield also contains the password mode of the file. This field is zero for files that are not password protected. The BDOS Search and Search Next functions can be used to access SFCBs directly. In addition, BDOS Function 102 can return the file date and time stamps and password mode for a specified file. Refer to Section 3, function 102, for a description of the format of a date and time stamp field.

CP/M 3 supports three types of file stamping: create, access, and update. Create stamps record when the file was created, access stamps record when the file was last opened, and update stamps record the last time the file was modified. Create and access stamps share the same field. As a result, file access stamps overwrite any create stamps.

The CP/M 3 utility, INITDIR, initializes a directory for date and time stamping by placing SFCBs in every fourth directory entry. Date and time stamping is not supported on disks that have not been initialized in this manner. For initialized disks the disks' Directory Label determines the type of date and time stamping supported for files on the drive. If a disk does not have a Directory Label, or if it is Read-Only, or if the disk's Directory Label does not specify date and time stamping, then date and time stamping for files is not performed. Note that the Directory Label is also time stamped, but these stamps are not made in an SFCB. Time stamp fields in the last eight bytes of the Directory Label record when it was created and last updated. Access stamping for Directory Labels is not supported.

The BDOS file system uses the CP/M 3 system date and time when it records a date and time stamp. This value is maintained in a field in the System Control Block (SCB). On CP/M 3 systems that support a hardware clock, the BIOS module directly updates the SCB system date and time field. Otherwise, date and time stamps record the last initialized value for the system date and time. The CP/M 3 DATE utility can be used to set the system date and time.

### 2.3.9 Record Blocking and Deblocking

Under CP/M 3, the logical record size for disk I/O is 128 bytes. This is the basic unit of data transfer between the operating system and transient programs. However, on disk, the record size is not restricted to 128 bytes. These records, called physical records, can range from 128 bytes to 4K bytes in size. Record blocking and deblocking is required on systems that support drives with physical record sizes larger than 128 bytes.

The process of building up physical records from 128 byte logical records is called record blocking. This process is required in write operations. The reverse process of breaking up physical records into their component 128 byte logical records is called record deblocking. This process is required in read operations. Under CP/M 3, record blocking and deblocking is normally performed by the BDOS.

Record deblocking implies a read-ahead operation. For example, if a transient program makes a BDOS function call to read a logical record that resides at the beginning of a physical record, the entire physical record is read into an internal buffer. Subsequent BDOS read calls for the remaining logical records access the buffer instead of the disk. Conversely, record blocking results in the postponement of physical write operations but only for data write operations. For example, if a transient program makes a BDOS write call, the logical record is placed in a buffer equal in size to the physical record size. The write operation on the physical record buffer is postponed until the buffer is needed in another I/O operation. Note that under CP/M 3, directory write operations are never postponed.

Postponing physical record write operations has implications for some applications programs. For those programs that involve file updating, it is often critical to guarantee that the state of the file on disk parallels the state of the file in memory after the update operation. This is only an issue on systems where physical write operations are postponed because of record blocking and deblocking. If the system should crash while a physical buffer is pending, data would be lost. To prevent this loss of data, the BDOS Flush Buffers function, function 48, can be called to force the write of any pending physical buffers.

**Note:** the CCP automatically discards all pending physical data buffers when it receives control following a system warm start. However, the BDOS file system automatically makes a Flush Buffers call in the Close File function. Thus, it is sufficient to close a file to ensure that all pending physical buffers for that file are written to the disk.

### 2.3.10 Multi-Sector I/O

CP/M 3 can read or write multiple 128-byte records in a single BDOS function call. This process, called multi-sector I/O, is useful primarily in sequential read and write operations, particularly on drives with physical record sizes larger than 128 bytes. In a multi-sector I/O operation, the BDOS file system bypasses, when possible, all intermediate record buffering. Data is transferred directly between the TPA and the drive. In addition, the BDOS informs the BIOS when it is reading or writing multiple physical records in sequence on a drive. The BIOS can use this information to further optimize the I/O operation resulting in even better performance. Thus, the primary objective of multi-sector I/O is to improve sequential I/O performance. The actual improvement obtained, however, depends on the hardware environment of the host system, and the implementation of the BIOS.

The number of records that can be supported with multi-sector I/O ranges from 1 to 128. This value can be set by BDOS function 44, Set multi-sector Count. The multi-sector count is set to one when a transient program begins execution. However, the CP/M 3 LOADER module executes with the multi-sector Count set to 128 unless the available TPA space is less than 16K. In addition, the CP/M 3 PIP utility also sets the multi-sector count to 128 when sufficient buffer space is available. Note that the greatest potential performance increases are obtained when the multi-sector count is set to 128. Of course, this requires a 16K buffer.

The multi-sector count determines the number of operations to be performed by the following BDOS functions:

- Sequential Read and Write functions
- Random Read and Write functions including Write Random with Zero Fill

If the multi-sector count is N, calling one of the above functions is equivalent to making N function calls. If a multi-sector I/O operation is interrupted with an error such as reading unwritten data, the file system returns in register H the number of 128-byte records successfully processed.

### 2.3.11 Disk Reset and Removable Media

The BDOS functions, Disk Reset (function 13) and Reset Drive (function 37) allow a program to control when a disk's directory is to be reinitialized for file operations. This process of initializing a disk's directory is called logging-in the drive. When CP/M 3 is cold started, all drives are in the reset state. Subsequently, as drives are referenced, they are automatically logged-in by the file system. Once logged-in, a drive remains in the logged-in state until it is reset by BDOS function 13 or 37. Following the reset operation, the drive is again automatically logged-in by the file system when it is next used. Note that BDOS functions 13 and 37 have similar effects except that function 13 is directed to all drives on the system. Any combination of drives can be reset with Function 37.

Logging-in a drive consists of several steps. The most important step is the initialization of the drive's allocation vector. The allocation vector records the allocation and deallocation of data blocks to files, as files are created, extended, deleted, and truncated. Another function performed during drive log-in is the initialization of the directory check-sum vector. The file system uses the check-sum vector to detect media changes on a drive. Note that permanent drives, which are drives that do not support media changes, might not have check-sum vectors. If directory hashing has been specified for the drive, a BIOS and GENCPM option, the file system creates a hash table for the directory during log-in.

The primary use of the drive reset functions is to prepare for a media change on a drive. Subsequently, when the drive is accessed by a BDOS function call, the drive is automatically logged-in. Resetting a drive has two important side effects. First of all, any pending blocking/deblocking buffers on the reset drive are discarded. Secondly, any data blocks that have been allocated to files that have not been closed are lost. An application program should close files, particularly files that have been written to, prior to resetting a drive.

Although CP/M 3 automatically relogs in removable media when media changes are detected, the application program should still explicitly reset a drive before prompting the user to change disks.

### 2.3.12 File Byte Counts

Although the logical record size of CP/M 3 is restricted to 128 bytes, CP/M 3 does provide a mechanism to store and retrieve a byte count for a file. This facility can identify the last byte of the last record of a file. The BDOS Compute File Size function returns the random record number, plus 1, of the last record of a file.

The BDOS Set File Attributes function can set a file's byte count. Conversely, the Open function can return a file's byte count to the cr field of the FCB. The BDOS Search and Search Next functions also return a file's byte count. These functions return the byte count in the s1 field of the FCB returned in the current DMA buffer (see BDOS Functions Returned 17 and 26).

Note that the file system does not access or update the byte count value in file read or write operations. However, the BDOS Make File function does set the byte count of a file to zero when it creates a file in the directory.

### 2.3.13 BDOS Error Handling

The BDOS file system responds to error situations in one of three ways:

Method 1. It returns to the calling program with return codes in register A, H, and L identifying the error.

Method 2. It displays an error message on the console, and branches to the BIOS warm start entry point, thereby terminating execution of the calling program.

Method 3. It displays an error message on the console, and returns to the calling program as in method 1.

The file system handles the majority of errors it detects by method 1. Two examples of this kind of error are the file not found error for the open function and the reading unwritten data error for a read function. More serious errors, such as disk I/O errors, are usually handled by method 2. Errors in this category, called physical and extended errors, can also be reported by methods 1 and 3 under program control.

The BDOS Error Mode, which can exist in three states, determines how the file system handles physical and extended errors. In the default state, the BDOS displays the error message, and terminates the calling program, method 2. In return error mode, the BDOS returns control to the calling program with the error identified in registers A, H, and L, method 1. In return and display mode, the BDOS returns control to the calling program with the error identified in registers A, H, and L, and also displays the error message at the console, method 3. While both return modes protect a program from termination because of a physical or extended error, the return and display mode also allows the calling program to take advantage of the built-in error reporting of the BDOS file system. Physical and extended errors are displayed on the console in the following format:

CP/M Error on d: error message
BDOS function = nn File = filename.typ

where d identifies the drive selected when the error condition is detected; error message identifies the error; nn is the BDOS function number, and filename.typ identifies the file specified by the BDOS function. If the BDOS function did not involve an FCB, the file information is omitted. Note that the second line of the above error message is displayed only in the banked version of CP/M 3 if expanded error message reporting is requested in GENCPM. It is not displayed in the nonbanked version of CP/M 3.

The BDOS physical errors are identified by the following error messages:

- Disk I/O
- Invalid Drive
- Read-Only File
- Read-Only Disk

The Disk I/O error results from an error condition returned to the BDOS from the BIOS module. The file system makes BIOS read and write calls to execute file-related BDOS calls. If the BIOS read or write routine detects an error, it returns an error code to the BDOS resulting in this error.

The Invalid Drive error also results from an error condition returned to the BDOS from the BIOS module. The BDOS makes a BIOS Select Disk call prior to accessing a drive to perform a requested BDOS function. If the BIOS does not support the selected disk, the BDOS returns an error code resulting in this error message.

The Read-Only File error is returned when a program attempts to write to a file that is marked with the Read-Only attribute. It is also returned to a program that attempts to write to a system file opened under user zero from a nonzero user number. In addition, this error is returned when a program attempts to write to a file password protected in Write mode if the program does not supply the correct password.

The Read-Only Disk error is returned when a program writes to a disk that is in read-only status. A drive can be placed in read-only status explicitly with the BDOS Write Protect Disk function.

The BDOS extended errors are identified by the following error messages:

- Password Error
- File Exists
- ? in Filename

The File Password error is returned when the file password is not supplied, or when it is incorrect. This error is reported only by the banked version of CP/M 3.

The File Exists error is returned by the BDOS Make File and Rename File functions when the BDOS detects a conflict such as a duplicate filename and type.

The ? in Filename error is returned when the BDOS detects a ? in the filename or type field of the passed FCB for the BDOS Rename File, Set File Attributes, Open File, Make File, and Truncate File functions.

The following paragraphs describe the error return code conventions of the BDOS file system functions. Most BDOS file system functions fall into three categories in regard to return codes: they return an Error Code, a Directory Code, or an Error Flag. The error conventions of CP/M 3 are designed to allow programs written for earlier versions of CP/M to run without modification.

The following BDOS functions return an Error Code in register A.

20. Read Sequential
21. Write Sequential
33. Read Random
34. Write Random
40. Write Random w/Zero Fill

The Error Code definitions for register A are shown in Table 2-7.

**Table 2-7. Register A BDOS Error Codes**

| *Code* | *Meaning* |
|---|---|
| 00 : | Function successful |
| 255 : | Physical error : refer to register H |
| 01 : | Reading unwritten data or no available directory space (Write Sequential) |
| 02 : | No available data block |
| 03 : | Cannot close current extent |
| 04 : | Seek to unwritten extent |
| 05 : | No available directory space |
| 06 : | Random record number out of range |
| 09 : | Invalid FCB (previous BDOS close call returned an error code and invalidated the FCB) |
| 10 : | Media Changed (A media change was detected on the FCB's drive after the FCB was opened) |

For BDOS read or write functions, the file system also sets register H when the returned Error Code is a value other than zero or 255. In this case, register H contains the number of 128-byte records successfully read or written before the error was encountered. Note that register H can contain only a nonzero value if the calling program has set the BDOS Multi-Sector Count to a value other than one; otherwise register H is set to zero. On successful functions, Error Code = 0, register H is also set to zero. If the Error Code equals 255, register H contains a physical error code (see Table 2-11).

The following BDOS functions return a Directory Code in register A:

15. Open File
16. Close File
17. Search For First
18. Search For Next
19. Delete File
22. Make File
23. Rename File
30. Set File Attributes
35. Compute File Size
99. Truncate File

* 100. Set Directory Label
102. Read File Date Stamps and Password Mode
** 103. Write File XFCB

* - This function is supported in the DIRLBL.RSX in the nonbanked version of CP/M 3.
** - This function is supported only in the banked version of CP/M 3.

The Directory Code definitions for register A are shown in Table 2-8.

**Table 2-8. BDOS Directory Codes**

| *Code* | *Meaning* |
|---|---|
| 00 — 03: | successful function |
| 255 : | unsuccessful function |

With the exception of the BDOS search functions, all functions in this category return with the directory code set to zero on successful returns. However, for the search functions, a successful Directory Code also identifies the relative starting position of the directory entry in the calling program's current DMA buffer.

If the Set BDOS Error Mode function is used to place the BDOS in return error mode, the following functions return an Error Flag on physical errors:

14. Select Disk
46. Get Disk Free Space
48. Flush Buffers
98. Free Blocks
101. Return Directory Label Data

The Error Flag definition for register A is shown in Table 2-9.

**Table 2-9. BDOS Error Flags**

| *Code* | *Meaning* |
|---|---|
| 00 : | successful function |
| 255 : | physical error : refer to register H |

The BDOS returns nonzero values in register H to identify a physical or extended error if the BDOS Error Mode is in one of the return modes. Except for functions that return a Directory Code, register A equal to 255 indicates that register H identifies the physical or extended error. For functions that return a Directory Code, if register A equals 255, and register H is not equal to zero, register H identifies the physical or extended error. Table 2-10 shows the physical and extended error codes returned in register H.

Table 2-10. BDOS Physical and Extended Errors

| Code | Meaning |
|---|---|
| 00 — | no error, or not a register H error |
| 01 — | Disk I/O error |
| 02 — | Read-Only Disk |
| 03 — | Read-Only File or File Opened under user zero from another user number or file password protected in write mode and correct password not specified. |
| 04 — | Invalid Drive : drive select error |
| 07 — | Password Error |
| 08 — | File Exists |
| 09 — | ? in Filename |

The following two functions represent a special case because they return an address in registers H and L.

27. Get Addr(Alloc)
31. Get Addr(Disk Parms)

When the BDOS is in return error mode, and it detects a physical error for these functions, it returns to the calling program with registers A, H, and L all set to 255. Otherwise, they return no error code.

## 2.4 Page Zero Initialization

Page Zero is the region of memory located from 0000H to 00FFH. This region contains several segments of code and data that are used by transient programs while running under CP/M 3. The code and data areas are shown in Table 2-11 for reference.

**Table 2-11. Page Zero Areas**

| *Location* From | To | *Contents* |
|---|---|---|
| 0000H — | 0002H | Contains a jump instruction to the BIOS warm start entry point at BIOS_base + 3. The address at location 0001H can also be used to make direct BIOS calls to the BIOS console status, console input, console output, and list output primitive functions. |
| 0003H — | 0004H | (Reserved) |
| 0005H — | 0007H | Contains a jump instruction to the BDOS, the LOADER, or to the most recently added RSX, and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, and LHLD 0006H places the address field of the jump instruction in the HL register pair. This value, minus one, is the highest address of memory available to the transient program. |
| 0008H — | 003AH | Reserved interrupt locations for Restarts 1 - 7 |
| 003BH — | 004FH | (Not currently used - reserved) |
| 0050H | | Identifies the drive from which the transient program was loaded. A value of one to sixteen identifies drives A through P. |
| 0051H — | 0052H | Contains the address of the password field of the first command-tail operand in the default DMA buffer beginning at 0080H. The CCP sets this field to zero if no password for the first command-tail operand is specified. |
| 0053H | | Contains the length of the password field for the first command-tail operand. The CCP also sets this field to zero if no password for the first command tail is specified. |
| 0054H — | 0055H | Contains the address of the password field of the second command-tail operand in the default DMA buffer beginning at 0080H. The CCP sets this field to zero if no password for the second command-tail operand is specified. |

**Table 2-11. (continued)**

| *Location* *From* | *To* | *Contents* |
|---|---|---|
| 0056H | | Contains the length of the password field for the second command-tail operand. The CCP also sets this field to zero if no password for the second command tail is specified. |
| 0057H — | 005BH | (Not currently used - reserved) |
| 005CH — | 007BH | Default File Control Block, FCB, area 1 initialized by the CCP from the first command-tail operand of the command line, if it exists. |
| 006CH — | 007BH | Default File Control Block, FCB, area 2 initialized by the CCP from the second command-tail operand of the command line, if it exists.<br><br>**Note:** this area overlays the last 16 bytes of default FCB area 1. To use the information in this area, a transient program must copy it to another location before using FCB area 1. |
| 007CH | | Current record position of default FCB area 1. This field is used with default FCB area 1 in sequential record processing. |
| 007DH — | 007FH | Optional default random record position. This field is an extension of default FCB area 1 used in random record processing. |
| 0080H — | 00FFH | Default 128-byte disk buffer. This buffer is also filled with the command tail when the CCP loads a transient program. |

The CCP initializes Page Zero prior to initiating a transient program. The fields at 0050H and above are initialized from the command line invoking the transient program. The command line format was described in detail in Section 1.6.2. To summarize, a command line usually takes the form:

<command> <command tail>

where

| | | |
|---|---|---|
| <command> | => | <file spec> |
| <command tail> | => | (no command tail) |
| | => | <file spec> |
| | => | <file spec><delimiter><file spec> |
| <file spec> | => | {d:}filename{.type} {;password} |

The CCP initializes the command drive field at 0050H to the drive index, A = 1, ..., P = 16, of the drive from which the transient program was loaded.

The default FCB at 005CH is defined if a command tail is entered. Otherwise, the fields at 005CH, 0068H to 006BH are set to binary zeros, the fields from 005DH to 0067H are set to blanks. The fields at 0051H through 0053H are set if a password is specified for the first <file spec> of the command tail. If not, these fields are set to zero.

The default FCB at 006CH is defined if a second <file spec> exists in the command tail. Otherwise, the fields at 006CH, 0078H to 007BH are set to binary zeros, the fields from 005DH to 0067H are set to blanks. The fields at 0054H through 0056H are set if a password is specified for the second <file spec> of the command tail. If not, these fields are set to zero.

Transient programs often use the default FCB at 005CH for file operations. This FCB may even be used for random file access because the three bytes starting at 007DH are available for this purpose. However, a transient program must copy the contents of the default FCB at 006CH to another area before using the default FCB at 005CH, because an open operation for the default FCB at 005CH overwrites the FCB data at 006CH.

The default DMA address for transient programs is 0080H. The CCP also initializes this area to contain the command tail of the command line. The first position contains the number of characters in the command line, followed by the command line characters. The character following the last command tail character is set to binary zero. The command line characters are preceded by a leading blank and are translated to ASCII upper-case. Because the 128-byte region beginning at 0080H is the default DMA, the BDOS file system moves 128-byte records to this area with read operations and accesses 128-byte records from this area with write operations. The transient program must extract the command tail information from this buffer before performing file operations unless it explicitly changes the DMA address with the BDOS Set DMA Address function.

The Page Zero fields of 0051H through 0056H locate the password fields of the first two file specifications in the command tail if they exist. These fields are provided so that transient programs are not required to parse the command tail for password fields. However, the transient program must save the password, or change the DMA address before performing file operations.

The following example illustrates the initialization of the command line fields of Page Zero. Assuming the following command line is typed at the console:

```
D > A: PROGRAM B:FILE,TYPE;PASS C;FILE.TYPE;PASSWORD
```

A hexadecimal dump of 0050H to 00A5H would show the Page Zero initialization performed by the CCP.

```
0050H: 01 8D 00 04 9D 00 08 00 00 00 00 00 02 46 49 4C .............FIL
0060H: 45 20 20 20 20 54 59 50 00 00 00 00 03 46 49 4C E....TYP.....FIL
0070H: 45 20 20 20 20 54 59 50 00 00 00 00 00 00 00 00 E....TYP........
0080H: 24 20 42 3A 46 49 4C 45 2E 54 59 50 3B 50 41 53 . B:FILE.TYP;PAS
0090H: 53 20 43 3A 46 49 4C 45 2E 54 59 50 3B 50 41 53 S C:FILE.TYP;PAS
00A0H: 53 57 4F 52 44 00                               SWORD.
```

*End of Section 2*

# Section 3
# BDOS Function Calls

This section describes each CP/M 3 system function, including the parameters a program must pass when calling the function, and the values the function returns to the program. The functions are arranged numerically for easy reference. You should be familiar with the BDOS calling conventions and other concepts presented in Section 2 before referencing this section.

| BDOS FUNCTION 0: SYSTEM RESET |
|---|
| Entry Parameters:<br>Register C: 00H |

The System Reset function terminates the calling program and returns control to the CCP via a warm start sequence (see Section 1.3.2). Calling this function has the same effect as a jump to location 0000H of Page Zero.

Note that the disk subsystem is not reset by System Reset under CP/M 3. The calling program can pass a return code to the CCP by calling Function 108, Get/Set Program Return Code, prior to making a System Reset call or jumping to location 0000H.

| BDOS FUNCTION 1: CONSOLE INPUT |
|---|
| Entry Parameters:<br>Register C: 01H<br><br>Returned Value:<br>Register A: ASCII Character |

The Console Input function reads the next character from the logical console, CONIN:, to register A. Graphic characters, along with carriage return, line-feed, and backspace, CTRL-H, are echoed to the console. Tab characters, CTRL-I, are expanded in columns of 8 characters. CTRL-S, CTRL-Q, and CTRL-P are normally intercepted as described below. All other non-graphic characters are returned in register A but are not echoed to the console.

When the Console Mode is in the default state (see Section 2.2.1), Function 1 intercepts the stop scroll, CTRL-S, start scroll, CTRL-Q, and start/stop printer echo, CTRL-P, characters. Any characters that are typed following a CTRL-S and preceding a CTRL-Q are also intercepted. However, if start/stop scroll has been disabled by the Console Mode, the CTRL-S, CTRL-Q, and CTRL-P characters are not intercepted. Instead, they are returned in register A, but are not echoed to the console.

If printer echo has been invoked, all characters that are echoed to the console are also sent to the list device, LST:.

Function 1 does not return control to the calling program until a non-intercepted character is typed, thus suspending execution if a character is not ready.

## BDOS FUNCTION 2: CONSOLE OUTPUT

Entry Parameters:
Registers C: 02H
E: ASCII Character

The Console Output function sends the ASCII character from register E to the logical console device, CONOUT:. When the Console Mode is in the default state (see Section 2.2.1), Function 2 expands tab characters, CTRL-I, in columns of 8 characters, checks for stop scroll, CTRL-S, start scroll, CTRL-Q, and echoes characters to the logical list device, LST:, if printer echo, CTRL-P, has been invoked.

## BDOS FUNCTION 3: AUXILIARY INPUT

Entry Parameters:
Register C: 03H

Returned Value:
Register A: ASCII Character

The Auxiliary Input function reads the next character from the logical auxiliary input device, AUXIN:, into register A. Control does not return to the calling program until the character is read.

## BDOS FUNCTION 4: AUXILIARY OUTPUT

Entry Parameters:
Registers C: 04H
E: ASCII Character

The Auxiliary Output function sends the ASCII character from register E to the logical auxiliary output device, AUXOUT:.

| BDOS FUNCTION 5: LIST OUTPUT |
|---|
| Entry Parameters:<br>Registers C: 05H<br>E: ASCII Character |

The List Output function sends the ASCII character in register E to the logical list device, LST:.

| BDOS FUNCTION 6: DIRECT CONSOLE I/O |
|---|
| Entry Parameters:<br>Registers C: 06H<br>E: 0FFH (input/status) or<br>0FEH (status) or<br>0FDH (input) or<br>char (output) |
| Returned Value:<br>Register A: char or status (no value) |

CP/M 3 supports direct I/O to the logical console, CONIN:, for those specialized applications where unadorned console input and output is required. Use Direct Console I/O carefully because it bypasses all the normal control character functions. Programs that perform direct I/O through the BIOS under previous releases of CP/M should be changed to use direct I/O so that they can be fully supported under future releases of MP/M and CP/M.

A program calls Function 6 by passing one of four different values in register E. The values and their meanings are summarized in Table 3-1.

**Table 3-1. Function 6 Entry Parameters**

| *Register E value* | *Meaning* |
|---|---|
| 0FFH | Console input/status command returns an input character; if no character is ready, a value of zero is returned. |
| 0FEH | Console status command (On return, register A contains 00 if no character is ready; otherwise it contains FFH.) |
| 0FDH | Console input command, returns an input character; this function will suspend the calling process until a character is ready. |
| ASCII character | Function 6 assumes that register E contains a valid ASCII character and sends it to the console. |

| BDOS FUNCTION 7: AUXILIARY INPUT STATUS |
| --- |
| Entry Parameters:<br>Register C: 07H<br><br>Returned Value:<br>Register A: Auxiliary Input Status |

The Auxiliary Input Status function returns the value 0FFH in register A if a character is ready for input from the logical auxiliary input device, AUXIN:. If no character is ready for input, the value 00H is returned.

| BDOS FUNCTION 8: AUXILIARY OUTPUT STATUS |
|---|
| Entry Parameters:<br>Register C: 08H<br><br>Returned Value:<br>Register A: Auxiliary Output Status |

The Auxiliary Output Status function returns the value 0FFH in register A if the logical auxiliary output device, AUXOUT:, is ready to accept a character for output. If the device is not ready for output, the value 00H is returned.

## BDOS FUNCTION 9: PRINT STRING

Entry Parameters:
Registers C: 09H
DE: String Address

The Print String function sends the character string addressed by register pair DE to the logical console, CONOUT:, until it encounters a delimiter in the string. Usually the delimiter is a dollar sign, $, but it can be changed to any other value by Function 110, Get/Set Output Delimiter. If the Console Mode is in the default state (see Section 2.2.1), Function 9 expands tab characters, CTRL-I, in columns of 8 characters. It also checks for stop scroll, CTRL-S, start scroll, CTRL-Q, and echoes to the logical list device, LST:, if printer echo, CTRL-P, has been invoked.

## BDOS FUNCTION 10: READ CONSOLE BUFFER

Entry Parameters:
Registers C: 0AH
DE: Buffer Address

Returned Value:
Console Characters in Buffer

The Read Console Buffer function reads a line of edited console input from the logical console, CONIN:, to a buffer that register pair DE addresses. It terminates input and returns to the calling program when it encounters a return, CTRL-M, or a line feed, CTRL-J, character. Function 10 also discards all input characters after the input buffer is filled. In addition, it outputs a bell character, CTRL-G, to the console when it discards a character to signal the user that the buffer is full. The input buffer addressed by DE has the following format:

| DE: +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | ... | +n |
|---|---|---|---|---|---|---|---|---|---|---|
| mx | nc | c1 | c2 | c3 | c4 | c5 | c6 | c7 | ... | ?? |

where mx is the maximum number of characters which the buffer holds, and nc is the number of characters placed in the buffer. The characters entered by the operator follow the nc value. The value mx must be set prior to making a Function 10 call and may range in value from 1 to 255. Setting mx to zero is equivalent to setting mx to one. The value nc is returned to the calling program and may range from zero to mx. If nc < mx, then uninitialized positions follow the last character, denoted by ?? in the figure. Note that a terminating return or line feed character is not placed in the buffer and not included in the count nc.

If register pair DE is set to zero, Function 10 assumes that an initialized input buffer is located at the current DMA address (see Function 26, Set DMA Address). This allows a program to put a string on the screen for the user to edit. To initialize the input buffer, set characters c1 through cn to the initial value followed by a binary zero terminator.

When a program calls Function 10 with an initialized buffer, Function 10 operates as if the user had typed in the string. When Function 10 encounters the binary zero terminator, it accepts input from the console. At this point, the user can edit the initialized string or accept it as it is by pressing the RETURN key. However, if the initialized string contains a return, CTRL-M, or a linefeed, CTRL-J, character, Function 10 returns to the calling program without giving the user the opportunity to edit the string.

The level of console editing supported by Function 10 differs for the banked and nonbanked versions of CP/M 3. Refer to the *CP/M Plus (CP/M Version 3) Operating System User's Guide* for a detailed description of console editing. In the nonbanked version, Function 10 recognizes the edit control characters summarized in Table 3-2.

**Table 3-2. Edit Control Characters (Nonbanked CP/M 3)**

| *Character* | *Edit Control Function* |
|---|---|
| rub/del | Removes and echoes the last character; GENCPM can change this function to CTRL-H |
| CTRL-C | Reboots when at the beginning of line; the Console Mode can disable this function |
| CTRL-E | Causes physical end of line |
| CTRL-H | Backspaces one character position; GENCPM can change this function to rub/del |
| CTRL-J | (Line-feed) terminates input line |
| CTRL-M | (Return) terminates input line |
| CTRL-P | Echoes console output to the list device |
| CTRL-R | Retypes the current line after new line |
| CTRL-U | Removes current line after new line |
| CTRL-X | Backspaces to beginning of current line |

The banked version of CP/M 3 expands upon the editing provided in the nonbanked version. The functionality of the two versions is similar when the cursor is positioned at the end of the line. However, in the banked version, the user can move the cursor anywhere in the current line, insert characters, delete characters, and perform other editing functions. In addition, the banked version saves the previous command line; it can be recalled when the current line is empty. Table 3-3 summarizes the edit control characters supported by Function 10 in the banked version of CP/M 3.

**Table 3-3. Edit Control Characters (Banked CP/M 3)**

| *Character* | *Edit Control Function* |
|---|---|
| rub/del | Removes and echoes the last character if at the end of the line; otherwise deletes the character to the left of the current cursor position; GENCPM can change this function to CTRL-H. |
| CTRL-A | Moves cursor one character to the left. |
| CTRL-B | Moves cursor to the beginning of the line when not at the beginning; otherwise moves cursor to the end of the line. |
| CTRL-C | Reboots when at the beginning of line; the Console Mode can disable this function. |
| CTRL-E | Causes physical end-of-line; if the cursor is positioned in the middle of a line, the characters at and to the right of the cursor are displayed on the next line. |
| CTRL-F | Moves cursor one character to the right. |
| CTRL-G | Deletes the character at the current cursor position when in the middle of the line; has no effect when the cursor is at the end of the line. |
| CTRL-H | Backspaces one character position when positioned at the end of the line; otherwise deletes the character to the left of the cursor; GENCPM can change this function to rub/del. |

**Table 3-3. (continued)**

| *Character* | *Edit Control Function* |
|---|---|
| CTRL-J | (Line-feed) terminates input; the cursor can be positioned anywhere in the line; the entire input line is accepted; sets the previous line buffer to the input line. |
| CTRL-K | Deletes all characters to the right of the cursor along with the character at the cursor. |
| CTRL-M | (Return) terminates input; the cursor can be positioned anywhere in the line; the entire input line is accepted; sets the previous line buffer to the input line. |
| CTRL-P | Echoes console output to the list device. |
| CTRL-R | Retypes the characters to the left of the cursor on the new line. |
| CTRL-U | Updates the previous line buffer to contain the characters to the left of the cursor; deletes current line, and advances to new line. |
| CTRL-W | Recalls previous line if current line is empty; otherwise moves cursor to end-of-line. |
| CTRL-X | Deletes all characters to the left of the cursor. |

For banked systems, Function 10 uses the console width field defined in the System Control Block. If the console width is exceeded when the cursor is positioned at the end of the line, Function 10 automatically advances to the next line. The beginning of the line can be edited by entering a CTRL-R.

When a character is typed while the cursor is positioned in the middle of the line, the typed character is inserted into the line. Characters at and to the right of the cursor are shifted to the right. If the console width is exceeded, the characters disappear off the right of the screen. However, these characters are not lost. They reappear if characters are deleted out of the line, or if a CTRL-E is typed.

## BDOS FUNCTION 11: GET CONSOLE STATUS

Entry Parameters:
Register C: 0BH

Returned Value:
Register A: Console Status

The Get Console Status function checks to see if a character has been typed at the logical console, CONIN:. If the Console Mode is in the default state (see Section 2.2.1), Function 11 returns the value 01H in register A when a character is ready. If a character is not ready, it returns a value of 00H.

If the Console Mode is in CTRL-C Only Status mode, Function 11 returns the value 01H in register A only if a CTRL-C has been typed at the console.

| BDOS FUNCTION 12: RETURN VERSION NUMBER |
| --- |
| Entry Parameters:<br>Register C: 0CH<br><br>Returned Value:<br>Register HL: Version Number |

The Return Version Number function provides information that allows version independent programming. It returns a two-byte value in register pair HL: H contains 00H for CP/M and L contains 31H, the BDOS file system version number. Function 12 is useful for writing applications programs that must run on multiple versions of CP/M and MP/M.

## BDOS FUNCTION 13: RESET DISK SYSTEM

Entry Parameters:
Register C: 0DH

The Reset Disk System function restores the file system to a reset state where all the disk drives are set to read-write (see Functions 28 and 29), the default disk is set to drive A, and the default DMA address is reset to 0080H. This function can be used, for example, by an application program that requires disk changes during operation. Function 37, Reset Drive, can also be used for this purpose.

| BDOS FUNCTION 14: SELECT DISK |
|---|
| Entry Parameters:<br>Registers C: 0EH<br>E: Selected Disk<br><br>Returned Value:<br>Registers A: Error Flag<br>H: Physical Error |

The Select Disk function designates the disk drive named in register E as the default disk for subsequent BDOS file operations. Register E is set to 0 for drive A, 1 for drive B, and so on through 15 for drive P in a full 16-drive system. In addition, Function 14 logs in the designated drive if it is currently in the reset state. Logging-in a drive activates the drive's directory until the next disk system reset or drive reset operation.

FCBs that specify drive code zero (dr = 00H) automatically reference the currently selected default drive. FCBs with drive code values between 1 and 16, however, ignore the selected default drive and directly reference drives A through P.

Upon return, register A contains a zero if the select operation was successful. If a physical error was encountered, the select function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the select function returns to the calling program with register A set to 0FFH and register H set to one of the following physical error codes:

01 : Disk I/O Error
04 : Invalid drive

## BDOS FUNCTION 15: OPEN FILE

Entry Parameters:
  Registers C: 0FH
  DE: FCB Address

Returned Value:
  Registers A: Directory Code
  H: Physical or Extended Error

The Open File function activates the FCB for a file that exists in the disk directory under the currently active user number or user zero. The calling program passes the address of the FCB in register pair DE, with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and byte 12 specifying the extent. Usually, byte 12 of the FCB is initialized to zero.

If the file is password protected in Read mode, the correct password must be placed in the first eight bytes of the current DMA, or have been previously established as the default password (see Function 106). If the current record field of the FCB, cr, is set to 0FFH, Function 15 returns the byte count of the last record of the file in the cr field. You can set the last record byte count for a file with Function 30, Set File Attributes. Note that the current record field of the FCB, cr, must be zeroed by the calling program before beginning read or write operations if the file is to be accessed sequentially from the first record.

If the current user is non-zero, and the file to be opened does not exist under the current user number, the open function searches user zero for the file. If the file exists under user zero, and has the system attribute, t2', set, the file is opened under user zero. Write operations are not supported for a file that is opened under user zero in this manner.

If the open operation is successful, the user's FCB is activated for read and write operations. The relevant directory information is copied from the matching directory FCB into bytes d0 through dn of the FCB. If the file is opened under user zero when the current user number is not zero, interface attribute f8' is set to one in the user's FCB. In addition, if the referenced file is password protected in Write mode, and the correct password was not passed in the DMA, or did not match the default password, interface attribute f7' is set to one. Write operations are not supported for an activated FCB if interface attribute f7' or f8' is true.

When the open operation is successful, the open function also makes an Access date and time stamp for the opened file when the following conditions are satisfied: the referenced drive has a directory label that requests Access date and time stamping, and the FCB extent number field is zero.

Upon return, the Open File function returns a directory code in register A with the value 00H if the open was successful, or FFH, 255 decimal, if the file was not found. Register H is set to zero in both of these cases. If a physical or extended error was encountered, the Open File function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, the Open File function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical or extended error codes:

01 : Disk I/O Error
04 : Invalid drive error
07 : File password error
09 : ? in the FCB filename or filetype field

## BDOS FUNCTION 16: CLOSE FILE

Entry Parameters:
Registers C: 10H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Physical or Extended Error

The Close File function performs the inverse of the Open File function. The calling program passes the address of an FCB in register pair DE. The referenced FCB must have been previously activated by a successful Open or Make function call (see Functions 15 and 22). Interface attribute f5' specifies how the file is to be closed as shown below:

f5' = 0 - Permanent close (default mode)
f5' = 1 - Partial close

A permanent close operation indicates that the program has completed file operations on the file. A partial close operation updates the directory, but indicates that the file is to be maintained in the open state.

If the referenced FCB contains new information because of write operations to the FCB, the close function permanently records the new information in the referenced disk directory. Note that the FCB does not contain new information, and the directory update step is bypassed if only read or update operations have been made to the referenced FCB.

Upon return, the close function returns a directory code in register A with the value 00H if the close was successful, or FFH, 255 Decimal, if the file was not found. Register H is set to zero in both of these cases. If a physical or extended error is encountered, the close function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the close function returns to the calling program with register A set to 0FFH and register H set to one of the following physical error codes:

01 : Disk I/O error
02 : Read/only disk
04 : Invalid drive error

| BDOS FUNCTION 17: SEARCH FOR FIRST |
|---|
| Entry Parameters:<br>Registers C: 11H<br>DE: FCB Address<br><br>Returned Value:<br>Registers A: Directory Code<br>H: Physical Error |

The Search For First function scans the directory for a match with the FCB addressed by register pair DE. Two types of searches can be performed. For standard searches, the calling program initializes bytes 0 through 12 of the referenced FCB, with byte 0 specifying the drive directory to be searched, bytes 1 through 11 specifying the file or files to be searched for, and byte 12 specifying the extent. Usually byte 12 is set to zero. An ASCII question mark, 63 decimal, 3F hex, in any of the bytes 1 through 12 matches all entries on the directory in the corresponding position. This facility, called ambiguous reference, can be used to search for multiple files on the directory. When called in the standard mode, the Search function scans for the first file entry in the specified directory that matches the FCB, and belongs to the current user number.

The Search For First function also initializes the Search For Next function. After the Search function has located the first directory entry matching the referenced FCB, the Search For Next function can be called repeatedly to locate all remaining matching entries. In terms of execution sequence, however, the Search For Next call must either follow a Search For First or Search For Next call with no other intervening BDOS disk-related function calls.

If byte 0 of the referenced FCB is set to a question mark, the Search function ignores the remainder of the referenced FCB, and locates the first directory entry residing on the current default drive. All remaining directory entries can be located by making multiple Search For Next calls. This type of search operation is not usually made by application programs, but it does provide complete flexibility to scan all current directory values. Note that this type of search operation must be performed to access a drive's directory label (see Section 2.3.6).

Upon return, the Search function returns a Directory Code in register A with the value 0 to 3 if the search is successful, or 0FFH, 255 Decimal, if a matching directory entry is not found. Register H is set to zero in both of these cases. For successful searches, the current DMA is also filled with the directory record containing the matching entry, and the relative starting position is A * 32 (that is, rotate the A register left 5 bits, or ADD A five times). Although it is not usually required for application programs, the directory information can be extracted from the buffer at this position.

If the directory has been initialized for date and time stamping by INITDIR, then an SFCB resides in every fourth directory entry, and successful Directory Codes are restricted to the values 0 to 2. For successful searches, if the matching directory record is an extent zero entry, and if an SFCB resides at offset 96 within the current DMA, contents of (DMA Address + 96) = 21H, the SFCB contains the date and time stamp information, and password mode for the file. This information is located at the relative starting position of 97 + (A * 10) within the current DMA in the following format:

0 - 3 : Create or Access Date and Time Stamp Field
4 - 7 : Update Date and Time Stamp Field
8 : Password Mode Field

(Refer to Section 2.3.8 for more information on SFCBs.)

If a physical error is encountered, the Search function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the Search function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

01 : Disk I/O error
04 : Invalid drive error

| BDOS FUNCTION 18: SEARCH FOR NEXT |
|---|
| Entry Parameters:<br>Register C: 12H<br><br>Returned Value:<br>Registers A: Directory Code<br>H: Physical Error |

The Search For Next function is identical to the Search For First function, except that the directory scan continues from the last entry that was matched. Function 18 returns a Directory code in register A, analogous to Function 17.

**Note:** in execution sequence, a Function 18 call must follow either a Function 17 or another Function 18 call with no other intervening BDOS disk-related function calls.

| BDOS FUNCTION 19: DELETE FILE |
|---|
| Entry Parameters:<br>Registers C: 13H<br>DE: FCB Address<br><br>Returned Value:<br>Registers A: Directory Code<br>H: Extended or Physical Error |

The Delete File function removes files or XFCBs that match the FCB addressed in register pair DE. The filename and filetype can contain ambiguous references, that is, question marks in bytes f1 through t3, but the dr byte cannot be ambiguous, as it can in the Search and Search Next functions. Interface attribute f5' specifies the type of delete operation that is performed.

f5' = 0 - Standard Delete (default mode)
f5' = 1 - Delete only XFCBs

If any of the files that the referenced FCB specify are password protected, the correct password must be placed in the first eight bytes of the current DMA buffer, or have been previously established as the default password (see Function 106).

For standard delete operations, the Delete function removes all directory entries belonging to files that match the referenced FCB. All disk directory and data space owned by the deleted files is returned to free space, and becomes available for allocation to other files. Directory XFCBs that were owned by the deleted files are also removed from the directory. If interface attribute f5' of the FCB is set to 1, Function 19 deletes only the directory XFCBs that match the referenced FCB.

**Note:** if any of the files that match the input FCB specification fail the password check, or are Read-Only, then the Delete function does not delete any files or XFCBs. This applies to both types of delete operations.

In nonbanked systems, file passwords and XFCBs are not supported. Thus, if the Delete function is called with interface attribute f5' set to true, the Delete function performs no action but returns with register A set to zero.

Upon return, the Delete function returns a Directory Code in register A with the value 0 if the delete is successful, or 0FFH, 255 Decimal, if no file that matches the referenced FCB is found. Register H is set to zero in both of these cases. If a physical, or extended error is encountered, the Delete function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, the Delete function returns to the calling program with register A set to 0FFH and register H set to one of the following physical or extended error codes:

01 : Disk I/O error
02 : Read-Only disk
03 : Read-Only file
04 : Invalid drive error
07 : File password error

| BDOS FUNCTION 20: READ SEQUENTIAL | | |
|---|---|---|
| Entry Parameters: | | |
| Registers | C: | 14H |
| | DE: | FCB Address |
| Returned Value: | | |
| Registers | A: | Error Code |
| | H: | Physical Error |

The Read Sequential function reads the next 1 to 128 128-byte records from a file into memory beginning at the current DMA address. The BDOS Multi-Sector Count (see Function 44) determines the number of records to be read. The default is one record. The FCB addressed by register pair DE must have been previously activated by an Open or Make function call.

Function 20 reads each record from byte cr of the extent, then automatically increments the cr field to the next record position. If the cr field overflows, then the function automatically opens the next logical extent and resets the cr field to 0 in preparation for the next read operation. The calling program must set the cr field to 0 following the Open call if the intent is to read sequentially from the beginning of the file.

Upon return, the Read Sequential function sets register A to zero if the read operation is successful. Otherwise, register A contains an error code identifying the error as shown below:

01 : Reading unwritten data (end-of-file)
09 : Invalid FCB
10 : Media change occurred
255 : Physical Error; refer to register H

Error Code 01 is returned if no data exists at the next record position of the file. Usually, the no data situation is encountered at the end of a file. However, it can also occur if an attempt is made to read a data block that has not been previously written, or an extent which has not been created. These situations are usually restricted to files created or appended with the BDOS random write functions (see Functions 34 and 40).

Error Code 09 is returned if the FCB is invalidated by a previous BDOS close call that returns an error.

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS Open, or Make Call.

Error Code 255 is returned if a physical error is encountered and the BDOS error mode is Return Error mode, or Return and Display Error mode (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

01 : Disk I/O error
04 : Invalid drive error

On all error returns except for physical error returns, A = 255, Function 20 sets register H to the number of records successfully read before the error is encountered. This value can range from 0 to 127 depending on the current BDOS Multi-Sector Count. It is always set to zero when the Multi-Sector Count is equal to one.

## BDOS FUNCTION 21: WRITE SEQUENTIAL

Entry Parameters:
Registers C: 15H
DE: FCB Address

Returned Value:
Registers A: Error Code
H: Physical Error

The Write Sequential function writes 1 to 128 128-byte data records, beginning at the current DMA address into the file named by the FCB addressed in register pair DE. The BDOS Multi-Sector Count (see Function 44) determines the number of 128 byte records that are written. The default is one record. The referenced FCB must have been previously activated by a BDOS Open or Make function call.

Function 21 places the record into the file at the position indicated by the cr byte of the FCB, and then automatically increments the cr byte to the next record position. If the cr field overflows, the function automatically opens, or creates the next logical extent, and resets the cr field to 0 in preparation for the next write operation. If Function 21 is used to write to an existing file, then the newly written records overlay those already existing in the file. The calling program must set the cr field to 0 following an Open or Make call if the intent is to write sequentially from the beginning of the file.

Function 21 makes an Update date and time for the file if the following conditions are satisfied: the referenced drive has a directory label that requests date and time stamping, and the file has not already been stamped for update by a previous Make or Write function call.

Upon return, the Write Sequential function sets register A to zero if the write operation is successful. Otherwise, register A contains an error code identifying the error as shown below:

01 : No available directory space
02 : No available data block
09 : Invalid FCB
10 : Media change occurred
255 : Physical Error : refer to register H

Error Code 01 is returned when the write function attempts to create a new extent that requires a new directory entry, and no available directory entries exist on the selected disk drive.

Error Code 02 is returned when the write command attempts to allocate a new data block to the file, and no unallocated data blocks exist on the selected disk drive.

Error Code 09 is returned if the FCB is invalidated by a previous BDOS close call that returns an error.

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS Open or Make call.

Error Code 255 is returned if a physical error is encountered and the BDOS error mode is Return Error mode, or Return and Display Error mode (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

01 : Disk I/O error
02 : Read-Only disk
03 : Read-Only file or
File open from user 0 when
the current user number is non-zero or
File password protected in Write mode
04 : Invalid drive error

On all error returns, except for physical error returns, A = 255, Function 21 sets register H to the number of records successfully written before the error was encountered. This value can range from 0 to 127 depending on the current BDOS Multi-Sector Count. It is always set to zero when the Multi-Sector Count is set to one.

## BDOS FUNCTION 22: MAKE FILE

Entry Parameters:
Registers C: 16H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Physical or Extended Error

The Make File function creates a new directory entry for a file under the current user number. It also creates an XFCB for the file if the referenced drive has a directory label that enables password protection on the drive, and the calling program assigns a password to the file.

The calling program passes the address of the FCB in register pair DE, with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and byte 12 set to the extent number. Usually, byte 12 is set to zero. Byte 32 of the FCB, the cr field, must be initialized to zero, before or after the Make call, if the intent is to write sequentially from the beginning of the file.

Interface attribute f6' specifies whether a password is to be assigned to the created file.

f6' = 0 - Do not assign password (default)
f6' = 1 - Assign password to created file

When attribute f6' is set to 1, the calling program must place the password in the first 8 bytes of the current DMA buffer, and set byte 9 of the DMA buffer to the password mode (see Function 102). Note that the Make function only interrogates interface attribute f6' if passwords are activated on the referenced drive. In non-banked systems, file passwords are not supported, and attribute f6' is never interrogated.

The Make function returns with an error if the referenced FCB names a file that currently exists in the directory under the current user number.

If the Make function is successful, it activates the referenced FCB for file operations by opening the FCB, and initializes both the directory entry and the referenced FCB to an empty file. It also initializes all file attributes to zero. In addition, Function 22 makes a Creation date and time stamp for the file if the following conditions are satisfied: the referenced drive has a directory label that requests Creation date and time stamping and the FCB extent number field is equal to zero. Function 22 also makes an Update stamp if the directory label requests update stamping and the FCB extent field is equal to zero.

If the referenced drive contains a directory label that enables password protection, and if interface attribute f6' has been set to 1, the Make function creates an XFCB for the file. In addition, Function 22 also assigns the password, and password mode placed in the first nine bytes of the DMA, to the XFCB.

Upon return, the Make function returns a directory code in register A with the value 0 if the make operation is successful, or 0FFH, 255 decimal, if no directory space is available. Register H is set to zero in both of these cases. If a physical or extended error is encountered, the Make function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the Make function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical or extended error codes:

01 : Disk I/O error
02 : Read-Only disk
04 : Invalid drive error
08 : File already exists
09 : ? in filename or filetype field

## BDOS FUNCTION 23: RENAME FILE

Entry Parameters:
Registers C: 17H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Physical or Extended Error

The Rename function uses the FCB, addressed by register pair DE, to change all directory entries of the file specified by the filename in the first 16 bytes of the FCB to the filename in the second 16 bytes. If the file specified by the first filename is password protected, the correct password must be placed in the first eight bytes of the current DMA buffer, or have been previously established as the default password (see Function 106). The calling program must also ensure that the filenames specified in the FCB are valid and unambiguous, and that the new filename does not already exist on the drive. Function 23 uses the dr code at byte 0 of the FCB to select the drive. The drive code at byte 16 of the FCB is ignored.

Upon return, the Rename function returns a Directory Code in register A with the value 0 if the rename is successful, or 0FFH, 255 Decimal, if the file named by the first filename in the FCB is not found. Register H is set to zero in both of these cases. If a physical or extended error is encountered, the Rename function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, the Rename function returns to the calling program with register A set to 0FFH and register H set to one of the following physical or extended error codes:

01 : Disk I/O error
02 : Read-Only disk
03 : Read-Only file
04 : Invalid drive error
07 : File password error
08 : File already exists
09 : ? in filename or filetype field

| BDOS FUNCTION 24: RETURN LOGIN VECTOR |
|---|
| Entry Parameters:<br>Register C: 18H<br><br>Returned Value:<br>Register HL: Login Vector |

Function 24 returns the login vector in register pair HL. The login vector is a 16-bit value with the least significant bit of L corresponding to drive A, and the high-order bit of H corresponding to the 16th drive, labelled P. A 0 bit indicates that the drive is not on-line, while a 1 bit indicates the drive is active. A drive is made active by either an explicit BDOS Select Disk call, number 14, or an implicit selection when a BDOS file operation specifies a non-zero dr byte in the FCB. Function 24 maintains compatibilty with earlier releases since registers A and L contain the same values upon return.

| BDOS FUNCTION 25: RETURN CURRENT DISK |
|---|
| Entry Parameters:<br>Register C: 19H<br><br>Returned Value:<br>Register A: Current Disk |

Function 25 returns the currently selected default disk number in register A. The disk numbers range from 0 through 15 corresponding to drives A through P.

## BDOS FUNCTION 26: SET DMA ADDRESS

Entry Parameters:
Registers C: 1AH
DE: DMA Address

DMA is an acronym for Direct Memory Address, which is often used in connection with disk controllers that directly access the memory of the computer to transfer data to and from the disk subsystem. Under CP/M 3, the current DMA is usually defined as the buffer in memory where a record resides before a disk write, and after a disk read operation. If the BDOS Multi-Sector Count is equal to one (see Function 44), the size of the buffer is 128 bytes. However, if the BDOS Multi-Sector Count is greater than one, the size of the buffer must equal N * 128, where N equals the Multi-Sector Count.

Some BDOS functions also use the current DMA to pass parameters, and to return values. For example, BDOS functions that check and assign file passwords require that the password be placed in the current DMA. As another example, Function 46, Get Disk Free Space, returns its results in the first 3 bytes of the current DMA. When the current DMA is used in this context, the size of the buffer in memory is determined by the specific requirements of the called function.

When a transient program is initiated by the CCP, its DMA address is set to 0080H. The BDOS Reset Disk System function, Function 13, also sets the DMA address to 0080H. The Set DMA function can change this default value to another memory address. The DMA address is set to the value passed in the register pair DE. The DMA address remains at this value until it is changed by another Set DMA Address, or Reset Disk System call.

## BDOS FUNCTION 27: GET ADDR(ALLOC)

Entry Parameters:
Register C: 1BH

Returned Value:
Register HL: ALLOC Address

CP/M 3 maintains an allocation vector in main memory for each active disk drive. Some programs use the information provided by the allocation vector to determine the amount of free data space on a drive. Note, however, that the allocation information might be inaccurate if the drive has been marked Read-Only.

Function 27 returns in register pair HL, the base address of the allocation vector for the currently selected drive. If a physical error is encountered when the BDOS error mode is one of the return modes (see Function 45), Function 27 returns the value 0FFFFH in the register pair HL.

In banked CP/M 3 systems, the allocation vector can be placed in bank zero. In this case, a transient program cannot access the allocation vector. However, the BDOS function, Get Disk Free Space (Function 46), can be used to directly return the number of free 128-byte records on a drive. The CP/M 3 utilities that display a drive's free space, DIR and SHOW, use Function 46 for that purpose.

| BDOS FUNCTION 28: WRITE PROTECT DISK |
|---|
| Entry Parameters:<br>Register C: 1CH |

The Write Protect Disk function provides temporary write protection for the currently selected disk by marking the drive as Read-Only. No program can write to a disk that is in the Read-Only state. A drive reset operation must be performed for a Read-Only drive to restore it to the Read-Write state (see Functions 13 and 37).

| BDOS FUNCTION 29: GET READ-ONLY VECTOR |
|---|
| Entry Parameters:<br>Register C: 1DH<br><br>Returned Value:<br>Register HL: R/O Vector Value |

Function 29 returns a bit vector in register pair HL that indicates which drives have the temporary Read-Only bit set. The Read-Only bit can be set only by a BDOS Write Protect Disk call.

The format of the bit vector is analagous to that of the login vector returned by Function 24. The least significant bit corresponds to drive A, while the most significant bit corresponds to drive P.

## BDOS FUNCTION 30: SET FILE ATTRIBUTES

Entry Parameters:
    Registers C: 1EH
    DE: FCB Address

Returned Value:
    Registers A: Directory Code
    H: Physical or Extended error

By calling the Set File Attributes function, a program can modify a file's attributes and set its last record byte count. Other BDOS functions can be called to interrogate these file parameters, but only Function 30 can change them. The file attributes that can be set or reset by Function 30 are f1' through f4', Read-Only, t1', System, t2', and Archive, t3'. The register pair DE addresses an FCB containing a filename with the appropriate attributes set or reset. The calling program must ensure that it does not specify an ambiguous filename. In addition, if the specified file is password protected, the correct password must be placed in the first eight bytes of the current DMA buffer or have been previously established as the default password (see Function 106).

Interface attribute f6' specifies whether the last record byte count of the specified file is to be set:

f6' = 0 - Do not set byte count (default mode)
f6' = 1 - Set byte count

If interface attribute f6' is set, the calling program must set the cr field of the referenced FCB to the byte count value. A program can access a file's byte count value with the BDOS Open, Search, or Search Next functions.

Function 30 searches the referenced directory for entries belonging to the current user number that matches the FCB specified name and type fields. The function then updates the directory to contain the selected indicators, and if interface attribute f6' is set, the specified byte count value. Note that the last record byte count is maintained in byte 13 of a file's directory FCBs.

File attributes t1', t2', and t3' are defined by CP/M 3. (They are described in Section 2.3.4.) Attributes f1' through f4' are not presently used, but can be useful for application programs, because they are not involved in the matching program used by the BDOS during Open File and Close File operations. Indicators f5' through f8' are reserved for use as interface attributes.

Upon return, Function 30 returns a Directory Code in register A with the value 0 if the function is successful, or 0FFH, 255 Decimal, if the file specified by the referenced FCB is not found. Register H is set to zero in both of these cases. If a physical or extended error is encountered, the Set File Attributes function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console, and the program is terminated. Otherwise, Function 30 returns to the calling program with register A set to 0FFH, and register H set to one of the following physical or extended error codes:

01 : Disk I/O error
02 : Read-Only disk
04 : Select error
07 : File password error
09 : ? in filename or filetype field

| BDOS FUNCTION 31: GET ADDR(DPB PARMS) |
|---|
| Entry Parameters:<br>Register C: 1FH<br><br>Returned Value:<br>Register HL: DPB Address |

Function 31 returns in register pair HL the address of the BIOS-resident Disk Parameter Block, DPB, for the currently selected drive. (Refer to the *CP/M Plus (CP/M Version 3) Operating System System Guide* for the format of the DPB). The calling program can use this address to extract the disk parameter values.

If a physical error is encountered when the BDOS error mode is one of the return modes (see Function 45), Function 31 returns the value 0FFFFH in the register pair HL.

| BDOS FUNCTION 32: SET/GET USER CODE | | |
|---|---|---|
| Entry Parameters: | | |
| Registers | C: | 20H |
| | E: | 0FFH (get) or User Code (set) |
| Returned Value: | | |
| Register | A: | Current Code or (no value) |

A program can change, or interrogate the currently active user number by calling Function 32. If register E = 0FFH, then the value of the current user number is returned in register A, where the value is in the range of 0 to 15. If register E is not 0FFH, then the current user number is changed to the value of E, modulo 16.

## BDOS FUNCTION 33: READ RANDOM

Entry Parameters:
Registers C: 21H
DE: FCB Address

Returned Value:
Registers A: Error Code
H: Physical Error

The Read Random function is similar to the Read Sequential function except that the read operation takes place at a particular random record number, selected by the 24-bit value constructed from the three byte, r0, r1, r2, field beginning at position 33 of the FCB. Note that the sequence of 24 bits is stored with the least significant byte first, r0, the middle byte next, r1, and the high byte last, r2. The random record number can range from 0 to 262,143. This corresponds to a maximum value of 3 in byte r2.

To read a file with Function 33, the calling program must first open the base extent, extent 0. This ensures that the FCB is properly initialized for subsequent random access operations. The base extent may or may not contain any allocated data. Function 33 reads the record specified by the random record field into the current DMA address. The function automatically sets the logical extent and current record values, but unlike the Read Sequential function, it does not advance the current record number. Thus, a subsequent Read Random call rereads the same record. After a random read operation, a file can be accessed sequentially, starting from the current randomly accessed position. However, the last randomly accessed record is reread or rewritten when switching from random to sequential mode.

If the BDOS Multi-Sector Count is greater than one (see Function 44), the Read Random function reads multiple consecutive records into memory beginning at the current DMA. The r0, r1, and r2 field of the FCB is automatically incremented to read each record. However, the FCBs random record number is restored to the first record's value upon return to the calling program.

Upon return, the Read Random function sets register A to zero if the read operation was successful. Otherwise, register A contains one of the following error codes:

01 : Reading unwritten data (end-of-file)
03 : Cannot close current extent
04 : Seek to unwritten extent
06 : Random record number out of range
10 : Media change occurred
255 : Physical Error : refer to register H

Error Code 01 is returned if no data exists at the next record position of the file. Usually, the no data situation is encountered at the end of a file. However, it can also occur if an attempt is made to read a data block that has not been previously written.

Error Code 03 is returned when the Read Random function cannot close the current extent prior to moving to a new extent.

Error Code 04 is returned when a read random operation accesses an extent that has not been created.

Error Code 06 is returned when byte 35, r2, of the referenced FCB is greater than 3.

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS Open or Make Call.

Error Code 255 is returned if a physical error is encountered, and the BDOS error mode is one of the return modes (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

01 : Disk I/O error
04 : Invalid drive error

On all error returns except for physical errors, A = 255, the Read Random function sets register H to the number of records successfully read before the error is encountered. This value can range from 0 to 127 depending on the current BDOS Multi-Sector Count. It is always set to zero when the Multi-Sector Count is equal to one.

## BDOS FUNCTION 34: WRITE RANDOM

Entry Parameters:
  Registers C: 22H
  DE: FCB Address

Returned Value:
  Registers A: Error Code
  H: Physical Error

The Write Random function is analogous to the Read Random function, except that data is written to the disk from the current DMA address. If the disk extent or data block where the data is to be written is not already allocated, the BDOS automatically performs the allocation before the write operation continues.

To write to a file using the Write Random function, the calling program must first open the base extent, extent 0. This ensures that the FCB is properly initialized for subsequent random access operations. If the file is empty, the calling program must create the base extent with the Make File function before calling Function 34. The base extent might or might not contain any allocated data, but it does record the file in the directory, so that the file can be displayed by the DIR utility.

The Write Random function sets the logical extent and current record positions to correspond with the random record being written, but does not change the random record number. Thus, sequential read or write operations can follow a random write, with the current record being reread or rewritten as the calling program switches from random to sequential mode.

Function 34 makes an Update date and time stamp for the file if the following conditions are satisfied: the referenced drive has a directory label that requests Update date and time stamping if the file has not already been stamped for update by a previous BDOS Make or Write call.

If the BDOS Multi-Sector Count is greater than one (see Function 44), the Write Random function reads multiple consecutive records into memory beginning at the current DMA. The r0, r1, and r2 field of the FCB is automatically incremented to write each record. However, the FCB's random record number is restored to the first record's value when it returns to the calling program. Upon return, the Write Random function sets register A to zero if the write operation is successful. Otherwise, register A contains one of the following error codes:

02 : No available data block
03 : Cannot Close current extent
05 : No available directory space
06 : Random record number out of range
10 : Media change occurred
255 : Physical Error : refer to register H

Error Code 02 is returned when the write command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.

Error Code 03 is returned when the Write Random function cannot close the current extent prior to moving to a new extent.

Error Code 05 is returned when the write function attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.

Error Code 06 is returned when byte 35, r2, of the referenced FCB is greater than 3.

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS Open or Make Call.

Error Code 255 is returned if a physical error is encountered and the BDOS error mode is one of the return modes (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, it is identified by register H as shown below:

01 : Disk I/O error
02 : Read-Only disk
03 : Read-Only file or
File open from user 0 when the current user number is nonzero or
File password protected in Write mode
04 : Invalid drive error

On all error returns, except for physical errors, A = 255, the Write Random function sets register H to the number of records successfully written before the error is encountered. This value can range from 0 to 127 depending on the current BDOS Multi-Sector Count. It is always set to zero when the Multi-Sector Count is equal to one.

## BDOS FUNCTION 35: COMPUTE FILE SIZE

Entry Parameters:
    Registers C: 23H
    DE: FCB Address

Returned Value:
    Registers A: Error Flag
    H: Physical or Extended error

    Random Record Field Set

The Compute File Size function determines the virtual file size, which is, in effect, the address of the record immediately following the end of the file. The virtual size of a file corresponds to the physical size if the file is written sequentially. If the file is written in random mode, gaps might exist in the allocation, and the file might contain fewer records than the indicated size. For example, if a single record with record number 262,143, the CP/M 3 maximum is written to a file using the Write Random function, then the virtual size of the file is 262,144 records even though only 1 data block is actually allocated.

To compute file size, the calling program passes in register pair DE the address of an FCB in random mode format, bytes r0, r1 and r2 present. Note that the FCB must contain an unambiguous filename and filetype. Function 35 sets the random record field of the FCB to the random record number + 1 of the last record in the file. If the r2 byte is set to 04, then the file contains the maximum record count 262,144.

A program can append data to the end of an existing file by calling Function 35 to set the random record position to the end of file, and then performing a sequence of random writes starting at the preset record address.

**Note:** the BDOS does not require that the file be open to use Function 35. However, if the file has been written to, it must be closed before calling Function 35. Otherwise, an incorrect file size might be returned.

Upon return, Function 35 returns a zero in register A if the file specified by the referenced FCB is found, or an 0FFH in register A if the file is not found. Register H is set to zero in both of these cases. If a physical error is encountered, Function 35 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, Function 35 returns to the calling program with register A set to 0FFH, and register H set to one of the following physical errors:

01 : Disk I/O error
04 : Invalid drive error

| BDOS FUNCTION 36: SET RANDOM RECORD |
|---|
| Entry Parameters:<br>Registers C: 24H<br>DE: FCB Address<br><br>Returned Value: Random Record Field Set |

The Set Random Record function returns the random record number of the next record to be accessed from a file that has been read or written sequentially to a particular point. This value is returned in the random record field, bytes r0, r1, and r2, of the FCB addressed by the register pair DE. Function 36 can be useful in two ways.

First, it is often necessary to initially read and scan a sequential file to extract the positions of various key fields. As each key is encountered, Function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record number minus one is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move directly to a particular record by performing a random read using the corresponding random record number that you saved earlier. The scheme is easily generalized when variable record lengths are involved, because the program need only store the buffer-relative byte position along with the key and record number to find the exact starting position of the keyed data at a later time.

A second use of Function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, then Function 36 is called to set the record number, and subsequent random read and write operations continue from the next record in the file.

## BDOS FUNCTION 37: RESET DRIVE

Entry Parameters:
Registers C: 25H
DE: Drive Vector

Returned Value:
Register A: 00H

The Reset Drive function programmatically restores specified drives to the reset state. A reset drive is not logged-in and is in Read-Write status. The passed parameter in register pair DE is a 16-bit vector of drives to be reset, where the least significant bit corresponds to the first drive A, and the high-order bit corresponds to the sixteenth drive, labelled P. Bit values of 1 indicate that the specified drive is to be reset.

| BDOS FUNCTION 38: ACCESS DRIVE |
|---|
| Entry Parameters:<br>Register C: 26H |

This is an MP/M function that is not supported under CP/M 3. If called, the file system returns a zero in register A indicating that the access request is successful.

| BDOS FUNCTION 39: FREE DRIVE |
| --- |
| Entry Parameters:<br>Register C: 27H |

This is an MP/M function that is not supported under CP/M 3. If called, the file system returns a zero in register A indicating that the free request is successful.

## BDOS FUNCTION 40: WRITE RANDOM WITH ZERO FILL

Entry Parameters:
Registers C: 28H
DE: FCB address

Returned Value:
Registers A: Error Code
H: Physical Error

The Write Random With Zero Fill function is identical to the Write Random function (Function 34) with the exception that a previously unallocated data block is filled with zeros before the record is written. If this function has been used to create a file, records accessed by a read random operation that contain all zeros identify unwritten random record numbers. Unwritten random records in allocated data blocks of files created using the Write Random function (Function 34) contain uninitialized data.

## BDOS FUNCTION 41: TEST AND WRITE RECORD

Entry Parameters:
    Registers C: 29H
    DE: FCB Address

Returned Value:
    Registers A: Error Code
    H: Physical Error

The Test and Write Record function is an MP/M II™ function that is not supported under CP/M 3. If called, Function 41 returns with register A set to 0FFH and register H set to zero.

## BDOS FUNCTION 42: LOCK RECORD

Entry Parameters:
Registers C: 2AH
DE: FCB Address

Returned Value:
Register A: 00H

The Lock Record function is an MP/M II function that is supported under CP/M 3 only to provide compatibility between CP/M 3 and MP/M. It is intended for use in situations where more than one running program has Read-Write access to a common file. Because CP/M 3 is a single-user operating system in which only one program can run at a time, this situation cannot occur. Thus, under CP/M 3, Function 42 performs no action except to return the value 00H in register A indicating that the record lock operation is successful.

BDOS FUNCTION 43: UNLOCK RECORD

Entry Parameters:
Registers C: 2BH
DE: FCB Address

Returned Value:
Register A: 00H

The Unlock Record function is an MP/M II function that is supported under CP/M 3 only to provide compatibility between CP/M 3 and MP/M. It is intended for use in situations where more than one running program has Read-Write access to a common file. Because CP/M 3 is a single-user operating system in which only one program can run at a time, this situation cannot occur. Thus, under CP/M 3, Function 43 performs no action except to return the value 00H in register A indicating that the record unlock operation is successful.

| BDOS FUNCTION 44: SET MULTI-SECTOR COUNT |
|---|
| Entry Parameters:<br>Registers C: 2CH<br>E: Number of Sectors<br><br>Returned Value:<br>Register A: Return Code |

The Set Multi-Sector Count function provides logical record blocking under CP/M 3. It enables a program to read and write from 1 to 128 records of 128 bytes at a time during subsequent BDOS Read and Write functions.

Function 44 sets the Multi-Sector Count value for the calling program to the value passed in register E. Once set, the specified Multi-Sector Count remains in effect until the calling program makes another Set Multi-Sector Count function call and changes the value. Note that the CCP sets the Multi-Sector Count to one when it initiates a transient program.

The Multi-Sector Count affects BDOS error reporting for the BDOS Read and Write functions. If an error interrupts these functions when the Multi-Sector is greater than one, they return the number of records successfully read or written in register H for all errors except for physical errors (A = 255).

Upon return, register A is set to zero if the specified value is in the range of 1 to 128. Otherwise, register A is set to 0FFH.

| BDOS FUNCTION 45: SET BDOS ERROR MODE |
|---|
| Entry Parameters:<br>Registers C: 2DH<br>E: BDOS Error Mode<br><br>Returned Value: None |

Function 45 sets the BDOS error mode for the calling program to the mode specified in register E. If register E is set to 0FFH, 255 decimal, the error mode is set to Return Error mode. If register E is set to 0FEH, 254 decimal, the error mode is set to Return and Display mode. If register E is set to any other value, the error mode is set to the default mode.

The SET BDOS Error Mode function determines how physical and extended errors (see Section 2.2.13) are handled for a program. The Error Mode can exist in three modes: the default mode, Return Error mode, and Return and Display Error mode. In the default mode, the BDOS displays a system message at the console that identifies the error and terminates the calling program. In the return modes, the BDOS sets register A to 0FFH, 255 decimal, places an error code that identifies the physical or extended error in register H and returns to the calling program. In Return and Display mode, the BDOS displays the system message before returning to the calling program. No system messages are displayed, however, when the BDOS is in Return Error mode.

| BDOS FUNCTION 46: GET DISK FREE SPACE | | |
|---|---|---|
| Entry Parameters: | | |
| Registers | C: | 2EH |
| | E: | Drive |
| Returned Value: | | First 3 bytes of current DMA buffer |
| Registers | A: | Error Flag |
| | H: | Physical Error |

The Get Disk Free Space function determines the number of free sectors, 128 byte records, on the specified drive. The calling program passes the drive number in register E, with 0 for drive A, 1 for B, and so on, through 15 for drive P in a full 16-drive system. Function 46 returns a binary number in the first 3 bytes of the current DMA buffer. This number is returned in the following format:

| fs0 | fs1 | fs2 |
|---|---|---|

Disk Free Space Field Format

fs0 = low byte
fs1 = middle byte
fs2 = high byte

Note that the returned free space value might be inaccurate if the drive has been marked Read-Only.

Upon return, register A is set to zero if the function is successful. However, if the BDOS Error Mode is one of the return modes (see Function 45), and a physical error is encountered, register A is set to 0FFH, 255 decimal, and register H is set to one of the following values:

01 - Disk I/O error
04 - Invalid drive error

## BDOS FUNCTION 47: CHAIN TO PROGRAM

Entry Parameters:
Registers C: 2FH
E: Chain Flag

The Chain To Program function provides a means of chaining from one program to the next without operator intervention. The calling program must place a command line terminated by a null byte, 00H, in the default DMA buffer. If register E is set to 0FFH, the CCP initializes the default drive and user number to the current program values when it passes control to the specified transient program. Otherwise, these parameters are set to the default CCP values. Note that Function 108, Get/Set Program Return Code, can be used to pass a two byte value to the chained program.

Function 47 does not return any values to the calling program and any encountered errors are handled by the CCP.

## BDOS FUNCTION 48: FLUSH BUFFERS

Entry Parameters:
    Registers C: 30H
              E: Purge Flag

Returned Value:
    Registers A: Error Flag
              H: Physical Error

The Flush Buffers function forces the write of any write-pending records contained in internal blocking/deblocking buffers. If register E is set to 0FFH, this function also purges all active data buffers. Programs that provide write with read verify support need to purge internal buffers to ensure that verifying reads actually access the disk instead of returning data that is resident in internal data buffers. The CP/M 3 PIP utility is an example of such a program.

Upon return, register A is set to zero if the flush operation is successful. If a physical error is encountered, the Flush Buffers function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, the Flush Buffers function returns to the calling program with register A set to 0FFH and register H set to the following physical error code:

01 : Disk I/O error
02 : Read/only disk
04 : Invalid drive error

| BDOS FUNCTION 49: | GET / SET SYSTEM CONTROL BLOCK |
|---|---|
| Entry Parameters: | |
| Registers C: | 31H |
| DE: | SCB PB Address |
| Returned Value: | |
| Registers A: | Returned Byte |
| HL: | Returned Word |

Function 49 allows access to parameters located in the CP/M 3 System Control Block (SCB). The SCB is a 100-byte data structure residing within the BDOS that contains flags and data used by the BDOS, CCP and other system components. Note that Function 49 is a CP/M 3 specific function. Programs intended for both MP/M II and CP/M 3 should either avoid the use of this function or isolate calls to this function in CP/M 3 version-dependent sections.

To use Function 49, the calling program passes the address of a data structure called the SCB parameter block in register pair DE. This data structure identifies the byte or word of the SCB to be updated or returned. The SCB parameter block is defined as:

```
SCBPB:      DB OFFSET     ; Offset within SCB
            DB SET        ; 0FFH if setting a byte
                          ; 0FEH if setting a word
                          ; 001H - 0FDH are reserved
                          ; 000H if a get operation
            DW VALUE      ; Byte or word value to be set
```

The OFFSET parameter identifies the offset of the field within the SCB to be updated or accessed. The SET parameter determines whether Function 49 is to set a byte or word value in the SCB or if it is to return a byte from the SCB. The VALUE parameter is used only in set calls. In addition, only the first byte of VALUE is referenced in set byte calls.

Use caution when you set SCB fields. Some of these parameters reflect the current state of the operating system. If they are set to invalid values, software errors can result. In general, do not use Function 49 to set a system parameter if another BDOS function can achieve the same result. For example, Function 49 can be called to update the Current DMA Address field within the SCB. This is not equivalent to making a Function 26, Set DMA Address call, and updating the SCB Current DMA field in this way would result in system errors. However, you can use Function 49 to return the Current DMA address. The System Control Block is summarized in the following table. Each of these fields is documented in detail in Appendix A.

**Table 3-4. System Control Block**

| *Offset* | *Description* |
|---|---|
| 00 — 04 | Reserved For System Use |
| 05 | BDOS version number |
| 06 — 09 | User Flags |
| 0A — 0F | Reserved For System Use |
| 10 — 11 | Program Error return code |
| 12 — 19 | Reserved For System Use |
| 1A | Console Width (columns) |
| 1B | Console Column Position |
| 1C | Console Page Length |
| 1D — 21 | Reserved For System Use |
| 22 — 23 | CONIN Redirection flag |
| 24 — 25 | CONOUT Redirection flag |
| 26 — 27 | AUXIN Redirection flag |
| 28 — 29 | AUXOUT Redirection flag |
| 2A — 2B | LSTOUT Redirection flag |
| 2C | Page Mode |
| 2D | Reserved For System Use |
| 2E | CTRL-H Active |
| 2F | Rubout Active |
| 30 — 32 | Reserved For System Use |
| 33 — 34 | Console Mode |
| 35 — 36 | Reserved For System Use |
| 37 | Output Delimiter |
| 38 | List Output Flag |
| 39 — 3B | Reserved For System Use |

Table 3-4. (continued)

| *Offset* | *Description* |
|---|---|
| 3C — 3D | Current DMA Address |
| 3E | Current Disk |
| 3F — 43 | Reserved For System Use |
| 44 | Current User Number |
| 45 — 49 | Reserved For System Use |
| 4A | BDOS Multi-Sector Count |
| 4B | BDOS Error Mode |
| 4C — 4F | Drive Search Chain (DISKS A:,E:,F:) |
| 50 | Temporary File Drive |
| 51 | Error Disk |
| 52 — 56 | Reserved For System Use |
| 57 | BDOS flags |
| 58 — 5C | Date Stamp |
| 5D — 5E | Common Memory Base Address |
| 5F — 63 | Reserved For System Use |

If Function 49 is called with the OFFSET parameter of the SCB parameter block greater than 63H, the function performs no action but returns with registers A and HL set to zero.

## BDOS FUNCTION 50: DIRECT BIOS CALLS

Entry Parameters:
Registers C: 32H
DE: BIOS PB Address

Returned Value: BIOS RETURN

Function 50 provides a direct BIOS call through the BDOS to the BIOS. The calling program passes the address of a data structure called the BIOS Parameter Block (BIOSPB) in register pair DE. The BIOSPB contains the BIOS function number and register contents as shown below:

```
BIOSPB:         db FUNC        ; BIOS function no.
                db AREG        ; A register contents
                dw BCREG       ; BC register contents
                dw DEREG       ; DE register contents
                dw HLREG       ; HL register contents
```

System Reset (Function 0) is equivalent to Function 50 with a BIOS function number of 1.

Note that the register pair BIOSPB fields (BCREG, DEREG, HLREG) are defined in low byte, high byte order. For example, in the BCREG field, the first byte contains the C register value, the second byte contains the B register value.

Under CP/M 3, direct BIOS calls via the BIOS jump vector are only supported for the BIOS Console I/O and List functions. You must use Function 50 to call any other BIOS functions. In addition, Function 50 intercepts BIOS Function 27 (Select Memory) calls and returns with register A set to zero. Refer to the *CP/M Plus (CP/M Version 3) Operating System System Guide* for the definition of the BIOS functions and their register passing and return conventions.

| BDOS FUNCTION 59: LOAD OVERLAY |
|---|
| Entry Parameters:<br>Registers C: 3BH<br>DE: FCB Address<br><br>Returned Value:<br>Registers A: Error Code<br>H: Physical Error |

Only transient programs with an RSX header can use the Load Overlay function because BDOS Function 59 is supported by the LOADER module. The calling program must have a header to force the LOADER to remain resident after the program is loaded (see Section 1.3).

Function 59 loads either an absolute or relocatable module. Relocatable modules are identified by a filetype of PRL. Function 59 does not call the loaded module.

The referenced FCB must be successfully opened before Function 59 is called. The load address is specified in the first two random record bytes of the FCB, r0 and r1. The LOADER returns an error if the load address is less than 100H, or if performing the requested load operation would overlay the LOADER, or any other Resident System Extensions that have been previously loaded.

When loading relocatable files, the LOADER requires enough room at the load address for the complete PRL file including the header and bit map (see Appendix B). Otherwise an error is returned. Function 59 also returns an error on PRL file load requests if the specified load address is not on a page boundary.

Upon return, Function 59 sets register A to zero if the load operation is successful. If the LOADER RSX is not resident in memory because the calling program did not have a RSX header, the BDOS returns with register A set to 0FFH and register H set to zero. If the LOADER detects an invalid load address, or if insufficient memory is available to load the overlay, Function 59 returns with register A set to 0FEH. All other error returns are consistent with the error codes returned by BDOS Function 20, Read Sequential.

## BDOS FUNCTION 60: CALL RESIDENT SYSTEM EXTENSION

Entry Parameters:
Registers C: 3CH
DE: RSX PB Address

Returned Value:
Registers A: Error Code
H: Physical Error

Function 60 is a special BDOS function that you use when you call Resident System Extensions. The RSX subfunction is specified in a structure called the RSX Parameter Block, defined as follows:

```
RSXPB:          db FUNC         ; RSX Function number
                db NUMPARMS     ; Number of word parameters
                dw PARMETER1    ; Parameter 1
                dw PARMETER2    ; Parameter 2
                . . .
                dw PARMETERn    ; Parameter n
```

RSX modules filter all BDOS calls and capture RSX function calls that they can handle. If there is no RSX module present in memory that can handle a specific RSX function call, the call is not trapped, and the BDOS returns 0FFH in registers A and L. RSX function numbers from 0 to 127 are available for CP/M 3 compatible software use. RSX function numbers 128 to 255 are reserved for system use.

| BDOS FUNCTION 98: FREE BLOCKS |
|---|
| Entry Parameters:<br>Register C: 62H<br><br>Returned Value:<br>Registers A: Error Flag<br>H: Physical Error |

The Free Blocks function scans all the currently logged-in drives, and for each drive returns to free space all temporarily-allocated data blocks. A temporarily-allocated data block is a block that has been allocated to a file by a BDOS write operation but has not been permanently recorded in the directory by a BDOS close operation. The CCP calls Function 98 when it receives control following a system warm start. Be sure to close your file, particularly any file you have written to, prior to calling Function 98.

In the nonbanked version of CP/M 3, Function 98 frees only temporarily allocated blocks for systems that request double allocation vectors in GENCPM.

Upon return, register A is set to zero if Function 98 is successful. If a physical error is encountered, the Free Blocks function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, the Free Blocks function returns to the calling program with register A set to 0FFH and register H set to the following physical error code:

04 : Invalid drive error

## BDOS FUNCTION 99: TRUNCATE FILE

Entry Parameters:
Registers C: 63H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Extended or Physical Error

The Truncate File function sets the last record of a file to the random record number contained in the referenced FCB. The calling program passes the address of the FCB in register pair DE, with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and bytes 33 through 35, r0, r1, and r2, specifying the last record number of the file. The last record number is a 24 bit value, stored with the least significant byte first, r0, the middle byte next, r1, and the high byte last, r2. This value can range from 0 to 262,143, which corresponds to a maximum value of 3 in byte r2.

If the file specified by the referenced FCB is password protected, the correct password must be placed in the first eight bytes of the current DMA buffer, or have been previously established as the default password (see Function 106).

Function 99 requires that the file specified by the FCB not be open, particularly if the file has been written to. In addition, any activated FCBs naming the file are not valid after Function 99 is called. Close your file before calling Function 99, and then reopen it after the call to continue processing on the file.

Function 99 also requires that the random record number field of the referenced FCB specify a value less than the current file size. In addition, if the file is sparse, the random record field must specify a record in a region of the file where data exists.

Upon return, the Truncate function returns a Directory Code in register A with the value 0 if the Truncate function is successful, or 0FFH, 255 decimal, if the file is not found or the record number is invalid. Register H is set to zero in both of these cases. If a physical or extended error is encountered, the Truncate function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, the Truncate function returns to the calling program with register A set to 0FFH and register H set to one of the following physical or extended error codes:

01 : Disk I/O error
02 : Read-Only disk
03 : Read-Only file
04 : Invalid drive error
07 : File password error
09 : ? in filename or filetype field

| BDOS FUNCTION 100: SET DIRECTORY LABEL |
|---|
| Entry Parameters:<br>Registers C: 64H<br>DE: FCB Address<br><br>Returned Value:<br>Registers A: Directory Code<br>H: Physical or Extended Error |

The Set Directory Label function creates a directory label, or updates the existing directory label for the specified drive. The calling program passes in register pair DE the address of an FCB containing the name, type, and extent fields to be assigned to the directory label. The name and type fields of the referenced FCB are not used to locate the directory label in the directory; they are simply copied into the updated or created directory label. The extent field of the FCB, byte 12, contains the user's specification of the directory label data byte. The definition of the directory label data byte is:

bit 7 - Require passwords for password-protected files
(Not supported in nonbanked CP/M 3 systems)
6 - Perform access date and time stamping
5 - Perform update date and time stamping
4 - Perform create date and time stamping
0 - Assign a new password to the directory label

If the current directory label is password protected, the correct password must be placed in the first eight bytes of the current DMA, or have been previously established as the default password (see Function 106). If bit 0, the low-order bit, of byte 12 of the FCB is set to 1, it indicates that a new password for the directory label has been placed in the second eight bytes of the current DMA.

Note that Function 100 is implemented as an RSX, DIRLBL.RSX, in nonbanked CP/M 3 systems. If Function 100 is called in nonbanked systems when the DIRLBL.RSX is not resident, an error code of 0FFH is returned.

Function 100 also requires that the referenced directory contain SFCBs to activate date and time stamping on the drive. If an attempt is made to activate date and time stamping when no SFCBs exist, Function 100 returns an error code of 0FFH in register A and performs no action. The CP/M 3 INITDIR utility initializes a directory for date and time stamping by placing an SFCB record in every fourth entry of the directory.

Function 100 returns a Directory Code in register A with the value 0 if the directory label create or update is successful, or 0FFH, 255 decimal, if no space exists in the referenced directory to create a directory label, or if date and time stamping was requested and the referenced directory did not contain SFCBs. Register H is set to zero in both of these cases. If a physical error or extended error is encountered, Function 100 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, Function 100 returns to the calling program with register A set to 0FFH and register H set to one of the following physical or extended error codes:

01 : Disk I/O error
02 : Read-Only disk
04 : Invalid drive error
07 : File password error

| BDOS FUNCTION 101: RETURN DIRECTORY LABEL DATA | | |
|---|---|---|
| Entry Parameters: | | |
| Registers | C: | 65H |
| | E: | Drive |
| Returned Value: | | |
| Registers | A: | Directory Label Data Byte |
| | H: | Physical Error |

The Return Directory Label Data function returns the data byte of the directory label for the specified drive. The calling program passes the drive number in register E with 0 for drive A, 1 for drive B, and so on through 15 for drive P in a full sixteen drive system. The format of the directory label data byte is shown below:

bit 7 - Require passwords for password protected files
6 - Perform access date and time stamping
5 - Perform update date and time stamping
4 - Perform create date and time stamping
0 - Directory label exists on drive

Function 101 returns the directory label data byte to the calling program in register A. Register A equal to zero indicates that no directory label exists on the specified drive. If a physical error is encountered by Function 101 when the BDOS Error mode is in one of the return modes (see Function 45), this function returns with register A set to 0FFH, 255 decimal, and register H set to one of the following:

01 : Disk I/O error
04 : Invalid drive error

| BDOS FUNCTION 102: READ FILE DATE STAMPS AND PASSWORD MODE |
|---|
| Entry Parameters:<br>Registers C: 66H<br>DE: FCB Address<br><br>Returned Value:<br>Registers A: Directory Code<br>H: Physical Error |

Function 102 returns the date and time stamp information and password mode for the specified file in byte 12 and bytes 24 through 32 of the specified FCB. The calling program passes in register pair DE, the address of an FCB in which the drive, filename, and filetype fields have been defined.

If Function 102 is successful, it sets the following fields in the referenced FCB:

```
byte 12 : Password mode field
          bit 7 - Read mode
          bit 6 - Write mode
          bit 4 - Delete mode
```

Byte 12 equal to zero indicates the file has not been assigned a password. In nonbanked systems, byte 12 is always set to zero.

```
byte 24 - 27 : Create or Access time stamp field
byte 28 - 31 : Update time stamp field
```

The date stamp fields are set to binary zeros if a stamp has not been made. The format of the time stamp fields is the same as the format of the date and time structure described in Function 104.

Upon return, Function 102 returns a Directory Code in register A with the value zero if the function is successful, or 0FFH, 255 decimal, if the specified file is not found. Register H is set to zero in both of these cases. If a physical or extended error is encountered, Function 102 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, Function 102 returns to the calling program with register A set to 0FFH and register H set to one of the following physical or extended error codes:

01 : Disk I/O error
04 : Invalid drive error
09 : ? in filename or filetype field

## BDOS FUNCTION 103: WRITE FILE XFCB

Entry Parameters:
Registers C: 67H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Physical Error

The Write File XFCB function creates a new XFCB or updates the existing XFCB for the specified file. The calling program passes in register pair DE the address of an FCB in which the drive, name, type, and extent fields have been defined. The extent field specifies the password mode and whether a new password is to be assigned to the file. The format of the extent byte is shown below:

FCB byte 12 (ex) : XFCB password mode
bit 7 - Read mode
bit 6 - Write mode
bit 5 - Delete mode
bit 0 - Assign new password to the file

If the specified file is currently password protected, the correct password must reside in the first eight bytes of the current DMA, or have been previously established as the default password (see Function 106). If bit 0 is set to 1, the new password must reside in the second eight bytes of the current DMA.

Upon return, Function 103 returns a Directory Code in register A with the value zero if the XFCB create or update is successful, or 0FFH, 255 decimal, if no directory label exists on the specified drive, or the file named in the FCB is not found, or no space exists in the directory to create an XFCB. Function 103 also returns with 0FFH in register A if passwords are not enabled by the referenced directory's label. On nonbanked systems, this function always returns with register A = 0FFH because passwords are not supported. Register H is set to zero in all of these cases. If a physical or extended error is encountered, Function 103 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, Function 103 returns to the calling program with register A set to 0FFH and register H set to one of the following physical or extended error codes:

01 : Disk I/O error
02 : Read-Only disk
04 : Invalid drive error
07 : File password error
09 : ? in filename or filetype field

| BDOS FUNCTION 104: SET DATE AND TIME |
|---|
| Entry Parameters:<br>Registers C: 68H<br>DE: DAT Address<br><br>Returned Value: none |

The Set Date and Time function sets the system internal date and time. The calling program passes the address of a 4-byte structure containing the date and time specification in the register pair DE. The format of the date and time (DAT) data structure is:

```
byte 0 - 1 : Date field
byte 2     : Hour field
byte 3     : Minute field
```

The date is represented as a 16-bit integer with day 1 corresponding to January 1, 1978. The time is represented as two bytes: hours and minutes are stored as two BCD digits.

This function also sets the seconds field of the system date and time to zero.

## BDOS FUNCTION 105: GET DATE AND TIME

Entry Parameters:
Registers C: 69H
DE: DAT Address

Returned Value:
Register A: seconds
DAT set

The Get Date and Time function obtains the system internal date and time. The calling program passes in register pair DE, the address of a 4-byte data structure which receives the date and time values. The format of the date and time, DAT, data structure is the same as the format described in Function 104. Function 105 also returns the seconds field of the system date and time in register A as a two digit BCD value.

## BDOS FUNCTION 106: SET DEFAULT PASSWORD

Entry Parameters:
Registers C: 6AH
DE: Password Address

Returned Value: none

The Set Default Password function allows a program to specify a password value before a file protected by the password is accessed. When the file system accesses a password-protected file, it checks the current DMA, and the default password for the correct value. If either value matches the file's password, full access to the file is allowed. Note that this function performs no action in nonbanked CP/M 3 systems because file passwords are not supported.

To make a Function 106 call, the calling program sets register pair DE to the address of an 8-byte field containing the password.

| BDOS FUNCTION 107: RETURN SERIAL NUMBER | |
|---|---|
| Entry Parameters: | |
| Registers C: | 6BH |
| DE: | Serial Number Field |
| Returned Value: | Serial number field set |

Function 107 returns the CP/M 3 serial number to the 6-byte field addressed by register pair DE.

| BDOS FUNCTION 108: GET/SET PROGRAM RETURN CODE | | |
|---|---|---|
| Entry Parameters: | | |
| Registers | C: | 6CH |
| | DE: | 0FFFFH (Get) or Program Return Code (Set) |
| Returned Value: | | |
| Register | HL: | Program Return Code or (no value) |

CP/M 3 allows programs to set a return code before terminating. This provides a mechanism for programs to pass an error code or value to a following job step in batch environments. For example, Program Return Codes are used by the CCP in CP/M 3's conditional command line batch facility. Conditional command lines are command lines that begin with a colon, :. The execution of a conditional command depends on the successful execution of the preceding command. The CCP tests the return code of a terminating program to determine whether it successfully completed or terminated in error. Program return codes can also be used by programs to pass an error code or value to a chained program (see Function 47, Chain To Program).

A program can set or interrogate the Program Return Code by calling Function 108. If register pair DE = 0FFFFH, then the current Program Return Code is returned in register pair HL. Otherwise, Function 108 sets the Program Return Code to the value contained in register pair DE. Program Return Codes are defined in Table 3-5.

**Table 3-5. Program Return Codes**

| *Code* | *Meaning* |
|---|---|
| 0000 — FEFF | Successful return |
| FF00 — FFFE | Unsuccessful return |
| 0000 | The CCP initializes the Program Return Code to zero unless the program is loaded as the result of program chain. |
| FF80 — FFFC | Reserved |
| FFFD | The program is terminated because of a fatal BDOS error. |
| FFFE | The program is terminated by the BDOS because the user typed a CTRL-C. |

| BDOS FUNCTION 109: GET/SET CONSOLE MODE |
|---|
| Entry Parameters:<br>Registers C: 6DH<br>DE: 0FFFFH (Get) or Console Mode (Set)<br><br>Returned Value:<br>Register HL: Console Mode or (no value) |

A program can set or interrogate the Console Mode by calling Function 109. If register pair DE = 0FFFFH, then the current Console Mode is returned in register HL. Otherwise, Function 109 sets the Console Mode to the value contained in register pair DE.

The Console Mode is a 16-bit system parameter that determines the action of certain BDOS Console I/O functions. The definition of the Console Mode is:

bit 0 = 1 - CTRL-C only status for Function 11.
= 0 - Normal status for Function 11.

bit 1 = 1 - Disable stop scroll, CTRL-S, start scroll, CTRL-Q, support.
= 0 - Enable stop scroll, start scroll support.

bit 2 = 1 - Raw console output mode. Disables tab expansion for Functions 2, 9 and 111. Also disables printer echo, CTRL-P, support.
= 0 - Normal console output mode.

bit 3 = 1 - Disable CTRL-C program termination
= 0 - Enable CTRL-C program termination

bits 8,9 - Console status mode for RSXs that perform console input redirection from a file. These bits determine how the RSX responds to console status requests.

bit 8 = 0, bit 9 = 0 - conditional status
bit 8 = 0, bit 9 = 1 - false status
bit 8 = 1, bit 9 = 0 - true status
bit 8 = 1, bit 9 = 1 - bypass redirection

Note that the Console Mode bits are numbered from right to left.

The CCP initializes the Console Mode to zero when it loads a program unless the program has an RSX that overrides the default value. Refer to Section 2.2.1 for detailed information on Console Mode.

## BDOS FUNCTION 110: GET/SET OUTPUT DELIMITER

Entry Parameters:
Registers C: 6EH
DE: 0FFFFH (Get) or
E: Output Delimiter (Set)

Returned Value:
Register A: Output Delimiter or (no value)

A program can set or interrogate the current Output Delimiter by calling Function 110. If register pair DE = 0FFFFH, then the current Output Delimiter is returned in register A. Otherwise, Function 110 sets the Output Delimiter to the value contained in register E.

Function 110 sets the string delimiter for Function 9, Print String. The default delimiter value is a dollar sign, $. The CCP restores the Output Delimiter to the default value when a transient program is loaded.

## BDOS FUNCTION 111: PRINT BLOCK

Entry Parameters:
Registers C: 6FH
DE: CCB Address

Returned Value: none

The Print Block function sends the character string located by the Character Control Block, CCB, addressed in register pair DE, to the logical console, CONOUT:. If the Console Mode is in the default state (see Section 2.2.1), Function 111 expands tab characters, CTRL-I, in columns of eight characters. It also checks for stop scroll, CTRL-S, start scroll, CTRL-Q, and echoes to the logical list device, LST:, if printer echo, CTRL-P, has been invoked.

The CCB format is:

byte 0 - 1 : Address of character string (word value)
byte 2 - 3 : Length of character string (word value)

| BDOS FUNCTION 112: LIST BLOCK |
|---|
| Entry Parameters:<br>Registers C: 70H<br>DE: CCB Address<br><br>Returned Value: none |

The List Block function sends the character string located by the Character Control Block, CCB, addressed in register pair DE, to the logical list device, LST:.

The CCB format is:

byte 0 - 1 : Address of character string (word value)
byte 2 - 3 : Length of character string (word value)

| BDOS FUNCTION 152: PARSE FILENAME |
| --- |
| Entry Parameters:<br>Registers C: 98H<br>DE: PFCB Address<br><br>Returned Value:<br>Register HL: Return code<br>Parsed file control block |

The Parse Filename function parses an ASCII file specification and prepares a File Control Block, FCB. The calling program passes the address of a data structure called the Parse Filename Control Block, PFCB, in register pair DE. The PFCB contains the address of the input ASCII filename string followed by the address of the target FCB as shown below:

```
PFCB: DW INPUT   ; Address of input ASCII string
      DW FCB     ; Address of target FCB
```

The maximum length of the input ASCII string to be parsed is 128 bytes. The target FCB must be 36 bytes in length.

Function 152 assumes the input string contains file specifications in the following form:

{d:}filename{.typ}{;password}

where items enclosed in curly brackets are optional. Function 152 also accepts isolated drive specifications d: in the input string. When it encounters one, it sets the filename, filetype, and password fields in the FCB to blank.

The Parse Filename function parses the first file specification it finds in the input string. The function first eliminates leading blanks and tabs. The function then assumes that the file specification ends on the first delimiter it encounters that is out of context with the specific field it is parsing. For instance, if it finds a colon, and it is not the second character of the file specification, the colon delimits the entire file specification.

Function 152 recognizes the following characters as delimiters:

space
tab
return
null
; (semicolon) - except before password field
= (equal)
< (less than)
> (greater than)
. (period) - except after filename and before filetype
: (colon) - except before filename and after drive
, (comma)
| (vertical bar)
[ (left square bracket)
] (right square bracket)

If Function 152 encounters a non-graphic character in the range 1 through 31 not listed above, it treats the character as an error. The Parse Filename function initializes the specified FCB shown in Table 3-6.

**Table 3-6. FCB Format**

| *Location* | *Contents* |
|---|---|
| byte 0 | The drive field is set to the specified drive. If the drive is not specified, the default drive code is used. 0 = default, 1 = A, 2 = B. |
| byte 1-8 | The name is set to the specified filename. All letters are converted to upper-case. If the name is not eight characters long, the remaining bytes in the filename field are padded with blanks. If the filename has an asterisk, *, all remaining bytes in the filename field are filled in with question marks, ?. An error occurs if the filename is more than eight bytes long. |
| byte 9-11 | The type is set to the specified filetype. If no filetype is specified, the type field is initialized to blanks. All letters are converted to upper-case. If the type is not three characters long, the remaining bytes in the filetype field are padded with blanks. If an asterisk, *, occurs, all remaining bytes are filled in with question marks, ?. An error occurs if the type field is more than three bytes long. |
| byte 12-15 | Filled in with zeros. |
| byte 16-23 | The password field is set to the specified password. If no password is specified, it is initialized to blanks. If the password is less than eight characters long, remaining bytes are padded with blanks. All letters are converted to upper-case. If the password field is more than eight bytes long, an error occurs. Note that a blank in the first position of the password field implies no password was specified. |
| byte 24-31 | Reserved for system use. |

If an error occurs, Function 152 returns an 0FFFFH in register pair HL.

On a successful parse, the Parse Filename function checks the next item in the input string. It skips over trailing blanks and tabs and looks at the next character. If the character is a null or carriage return, it returns a 0 indicating the end of the input string. If the character is a delimiter, it returns the address of the delimiter. If the character is not a delimiter, it returns the address of the first trailing blank or tab.

If the first non-blank or non-tab character in the input string is a null, 0, or carriage return, the Parse Filename function returns a zero indicating the end of string.

If the Parse Filename function is to be used to parse a subsequent file specification in the input string, the returned address must be advanced over the delimiter before placing it in the PFCB.

*End of Section 3*

# Section 4
# Programming Examples

The programs presented in this section illustrate how to use the BDOS functions described in the previous section. The examples show how to copy a file, how to dump a file, how to create or access a random access file, and how to write an RSX program.

## 4.1 A Sample File-To-File Copy Program

The following program illustrates simple file operations. You can create the program source file, COPY.ASM, using ED or another editor, and then assemble COPY.ASM using MAC™. MAC produces the file COPY.HEX. Use the utility HEXCOM to produce a COPY.COM file that can execute under CP/M 3.

The COPY program first sets the stack pointer to a local area, then moves the second name from the default area at 006CH to a 33-byte file control block named DFCB. The DFCB is then prepared for file operations by clearing the current record field. Because the CCP sets up the source FCB at 005CH upon entry to the COPY program, the source and destination FCBs are now ready for processing. To prepare the source FCB, the CCP places the first name into the default FCB, with the proper fields zeroed, including the current record field at 007CH.

COPY continues by opening the source file, deleting any existing destination file, and then creating the destination file. If each of these operations is successful, the COPY program loops at the label COPY until each record is read from the source file and placed into the destination file. Upon completion of the data transfer, the destination file is closed, and the program returns to the CCP command level by jumping to BOOT.

```
            ;       sample file-to-file copy program
            ;
            ;       at the ccp level, the command
            ;
            ;               copy a:x.y b:u.v
            ;
            ;       copies the file named x.y from drive
            ;       a to a file named u.v on drive b.
            ;
0000 =      boot    equ     0000h   ; system reboot
0005 =      bdos    equ     0005h   ; bdos entry point
005c =      fcb1    equ     005ch   ; first file name
005c =      sfcb    equ     fcb1    ; source fcb
006c =      fcb2    equ     006ch   ; second file name
0080 =      dbuff   equ     0080h   ; default buffer
0100 =      tpa     equ     0100h   ; beginning of tpa
            ;
0009 =      printf  equ     9       ; print buffer func#
000f =      openf   equ     15      ; open file func#
0010 =      closef  equ     16      ; close file func#
0013 =      deletef equ     19      ; delete file func#
0014 =      readf   equ     20      ; sequential read
0015 =      writef  equ     21      ; sequential write
0016 =      makef   equ     22      ; make file func#
            ;
0100                org     tpa     ; beginning of tpa
0100 311b02         lxi     sp,stack; local stack
            ;
            ;       move second file name to dfcb
0103 0e10           mvi     c,16    ; half an fcb
0105 116c00         lxi     d,fcb2  ; source of move
0108 21da01         lxi     h,dfcb  ; destination fcb
010b 1a     mfcb:   ldax    d       ; source fcb
010c 13             inx     d       ; ready next
010d 77             mov     m,a     ; dest fcb
010e 23             inx     h       ; ready next
010f 0d             dcr     c       ; count 16...0
0110 c20b01         jnz     mfcb    ; loop 16 times
            ;
            ;       name has been moved, zero cr
0113 af             xra     a       ; a = 00h
0114 32fa01         sta     dfcbcr  ; current rec = 0
            ;
```

```
              ;       source and destination fcbs ready
              ;
0117 115c00           lxi     d,sfcb  ; source file
011a cd6901           call    open    ; error if 255
011d 118701           lxi     d,nofile; ready message
0120 3c               inr     a       ; 255 becomes 0
0121 cc6101           cz      finis   ; done if no file
              ;
              ;       source file open, prep destination
0124 11da01           lxi     d,dfcb  ; destination
0127 cd7301           call    delete  ; remove if present
              ;
012a 11da01           lxi     d,dfcb  ; destination
012d cd8201           call    make    ; create the file
0130 119601           lxi     d,nodir ; ready message
0133 3c               inr     a       ; 255 becomes 0
0134 cc6101           cz      finis   ; done if no dir space
              ;
              ;       source file open, dest file open
              ;       copy until end of file on source
              ;
0137 115c00   copy:   lxi     d,sfcb  ; source
013a cd7801           call    read    ; read next record
013d b7               ora     a       ; end of file?
013e c25101           jnz     eofile  ; skip write if so
              ;
              ;       not end of file, write the record
0141 11da01           lxi     d,dfcb  ; destination
0144 cd7d01           call    write   ; write record
0147 11a901           lxi     d,space ; ready message
014a b7               ora     a       ; 00 if write ok
014b c46101           cnz     finis   ; end if so
014e c33701           jmp     copy    ; loop until eof
              ;
              eofile: ; end of file, close destination
0151 11da01           lxi     d,dfcb  ; destination
0154 cd6e01           call    close   ; 255 if error
0157 21bb01           lxi     h,wrprot; ready message
015a 3c               inr     a       ; 255 becomes 00
015b cc6101           cz      finis   ; should not happen
              ;
              ;       copy operation complete, end
015e 11cc01           lxi     d,normal; ready message
              ;
              finis:  ; write message given by de, reboot
0161 0e09             mvi     c,printf
0163 cd0500           call    bdos    ; write message
0166 c30000           jmp     boot    ; reboot system
              ;
```

```
            ;       system interface subroutines
            ;       (all return directly from bdos)
            ;
0169 0e0f   open:   mvi     c,openf
016b c30500         jmp     bdos
            ;
016e 0e10   close:  mvi     c,closef
0170 c30500         jmp     bdos
            ;
0173 0e13   delete: mvi     c,deletef
0175 c30500         jmp     bdos
            ;
0178 0e14   read:   mvi     c,readf
017a c30500         jmp     bdos
            ;
017d 0e15   write:  mvi     c,writef
017f c30500         jmp     bdos
            ;
0182 0e16   make:   mvi     c,makef
0184 c30500         jmp     bdos
            ;
            ;       console messages
0187 6e6f20fnofile: db      'no source file$'
0196 6e6f209nodir:  db      'no directory space$'
01a9 6f7574fspace:  db      'out of data space$'
01bb 7772695wrprot: db      'write protected?$'
01cc 636f700normal: db      'copy complete$'
            ;
            ;       data areas
01da        dfcb:   ds      33      ; destination fcb
01fa =      dfcbcr  equ     dfcb+32 ; current record
            ;
01fb                ds      32      ; 16 level stack
            stack:
021b                end
```

Note that this program makes several simplifications and could be enhanced. First, it does not check for invalid filenames that could, for example, contain ambiguous references. This situation could be detected by scanning the 32-byte default area starting at location 005CH for ASCII question marks. To check that the filenames have, in fact, been included, COPY could check locations 005DH and 006DH for nonblank ASCII characters. Finally, a check should be made to ensure that the source and destination filenames are different. Speed could be improved by buffering more data on each read operation. For example, you could determine the size of memory by fetching FBASE from location 0006H, and use the entire remaining portion of memory for a data buffer. You could also use CP/M 3's Multi-Sector I/O facility to read and write data in up to 16K units.

## 4.2 A Sample File Dump Utility

The following dump program reads an input file specified in the CCP command line, and then displays the content of each record in hexadecimal format at the console.

```
                ; DUMP program reads input file and displays hex data
                ;
0100                    org     100h
0005 =          bdos    equ     0005h   ;dos entry point
0001 =          cons    equ     1       ;read console
0002 =          typef   equ     2       ;type function
0009 =          printf  equ     9       ;buffer print entry
000b =          brkf    equ     11      ;break key function (true if char
000f =          openf   equ     15      ;file open
0014 =          readf   equ     20      ;read function
                ;
005c =          fcb     equ     5ch     ;file control block address
0080 =          buff    equ     80h     ;input disk buffer address
                ;
                ;       non graphic characters
000d =          cr      equ     0dh     ;carriage return
000a =          lf      equ     0ah     ;line feed
                ;
                ;       file control block definitions
005c =          fcbdn   equ     fcb+0   ;disk name
005d =          fcbfn   equ     fcb+1   ;file name
0065 =          fcbft   equ     fcb+9   ;disk file type (3 characters)
0068 =          fcbrl   equ     fcb+12  ;file's current reel number
006b =          fcbrc   equ     fcb+15  ;file's record count (0 to 128)
007c =          fcbcr   equ     fcb+32  ;current (next) record number (0
007d =          fcbln   equ     fcb+33  ;fcb length
                ;
                ;       set up stack
0100 210000             lxi     h,0
0103 39                 dad     sp
                ;       entry stack pointer in hl from the ccp
0104 221502             shld    oldsp
                ;       set sp to local stack area (restored at finis)
0107 315702             lxi     sp,stktop
                ;       read and print successive buffers
010a cdc101             call    setup   ;set up input file
010d feff               cpi     255     ;255 if file not present
010f c21b01             jnz     openok  ;skip if open is ok
                ;
```

```
              ;       file not there, give error message and return
0112 11f301           lxi     d,opnmsg
0115 cd9c01           call    err
0118 c35101           jmp     finis   ;to return
              ;
              openok: ;open operation ok, set buffer index to end
011b 3e80             mvi     a,80h
011d 321302           sta     ibp     ;set buffer pointer to 80h
              ;       hl contains next address to print
0120 210000           lxi     h,0     ;start with 0000
              ;
              gloop:
0123 e5               push    h       ;save line position
0124 cda201           call    gnb
0127 e1               pop     h       ;recall line position
0128 da5101           jc      finis   ;carry set by gnb if end file
012b 47               mov     b,a
              ;       print hex values
              ;       check for line fold
012c 7d               mov     a,l
012d e60f             ani     0fh     ;check low 4 bits
012f c24401           jnz     nonum
              ;       print line number
0132 cd7201           call    crlf
              ;
              ;       check for break key
0135 cd5901           call    break
              ;       accum lsb = 1 if character ready
0138 0f               rrc             ;into carry
0139 da5101           jc      finis   ;do not print any more
; 013c 7c             mov     a,h
013d cd8f01           call    phex
0140 7d               mov     a,l
0141 cd8f01           call    phex
              nonum:
0144 23               inx     h       ;to next line number
0145 3e20             mvi     a,' '
0147 cd6501           call    pchar
014a 78               mov     a,b
014b cd8f01           call    phex
014e c32301           jmp     gloop
```

```
                ;
                finis:
                ;       end of dump
0151 cd7201             call    crlf
0154 2a1502             lhld    oldsp
0157 f9                 sphl
                ;       stack pointer contains ccp's stack location
0158 c9                 ret             ;to the ccp
                ;
                ;
                ;       subroutines
                ;
                break:  ;check break key (actually any key will do)
0159 e5d5c5             push h! push d! push b; environment saved
015c 0e0b               mvi     c,brkf
015e cd0500             call    bdos
0161 c1d1e1             pop b! pop d! pop h; environment restored
0164 c9                 ret
                ;
                pchar:  ;print a character
0165 e5d5c5             push h! push d! push b; saved
0168 0e02               mvi     c,typef
016a 5f                 mov     e,a
016b cd0500             call    bdos
016e c1d1e1             pop b! pop d! pop h; restored
0171 c9                 ret
                ;
                crlf:
0172 3e0d               mvi     a,cr
0174 cd6501             call    pchar
0177 3e0a               mvi     a,lf
0179 cd6501             call    pchar
017c c9                 ret
                ;
                ;
                pnib:   ;print nibble in reg a
017d e60f               ani     0fh     ;low 4 bits
017f fe0a               cpi     10
0181 d28901             jnc     p10
                ;       less than or equal to 9
0184 c630               adi     '0'
0186 c38b01             jmp     prn
                ;
                ;       greater or equal to 10
0189 c637       p10:    adi     'a' - 10
018b cd6501     prn:    call    pchar
018e c9                 ret
                ;
```

```
              phex:    ;print hex char in reg a
018f f5                push    psw
0190 0f                rrc
0191 0f                rrc
0192 0f                rrc
0193 0f                rrc
0194 cd7d01            call    pnib    ;print nibble
0197 f1                pop     psw
0198 cd7d01            call    pnib
019b c9                ret
              ;
              err:     ;print error message
              ;        d,e addresses message ending with"$"
019c 0e09              mvi     c,printf        ;print buffer function
019e cd0500            call    bdos
01a1 c9                ret
              ;
              ;
              gnb:     ;get next byte
01a2 3a1302            lda     ibp
01a5 fe80              cpi     80h
01a7 c2b301            jnz     g0
              ;        read another buffer
              ;
              ;
01aa cdce01            call    diskr
01ad b7                ora     a       ;zero value if read ok
01ae cab301            jz      g0      ;for another byte
              ;        end of data, return with carry set for eof
01b1 37                stc
01b2 c9                ret
              ;
              g0:      ;read the byte at buff+reg a
01b3 5f                mov     e,a     ;ls byte of buffer index
01b4 1600              mvi     d,0     ;double precision index to de
01b6 3c                inr     a       ;index=index+1
01b7 321302            sta     ibp     ;back to memory
              ;        pointer is incremented
              ;        save the current file address
01ba 218000            lxi     h,buff
01bd 19                dad     d
              ;        absolute character address is in hl
01be 7e                mov     a,m
              ;        byte is in the accumulator
01bf b7                ora     a       ;reset carry bit
01c0 c9                ret
              ;
```

```
                setup:  ;set up file
                ;       open the file for input
01c1 af                 xra     a       ;zero to accum
01c2 327c00             sta     fcbcr   ;clear current record
                ;
01c5 115c00             lxi     d,fcb
01c8 0e0f               mvi     c,openf
01ca cd0500             call    bdos
                ;       255 in accum if open error
01cd c9                 ret
                ;
                diskr:  ;read disk file record
01ce e5d5c5             push h! push d! push b
01d1 115c00             lxi     d,fcb
01d4 0e14               mvi     c,readf
01d6 cd0500             call    bdos
01d9 c1d1e1             pop b! pop d! pop h
01dc c9                 ret
                ;
                ;       fixed message area
01dd 46494c0signon: db          'file dump version 2.0$'
01f3 0d0a4e0opnmss: db          cr,lf,'no input file present on disk$'

                ;       variable area
0213            ibp:    ds      2       ;input buffer pointer
0215            oldsp:  ds      2       ;entry sp value from ccp
                ;
                ;       stack area
0217                    ds      64      ;reserve 32 level stack
                stktop:
                ;
0257                    end
```

## 4.3 A Sample Random Access Program

This example is an extensive but complete example of random access operation. The following program reads or writes random records upon command from the terminal. When the program has been created, assembled, and placed into a file labeled RANDOM.COM, the CCP level command

```
A>RANDOM X.DAT
```

can start the test program. In this case, the RANDOM program looks for a file X.DAT and, if it finds it, prompts the console for input. If X.DAT is not found, RANDOM creates the file before displaying the prompt. Each prompt takes the form:

next command?

and is followed by operator input, terminated by a carriage return. The input commands take the form:

nW nR nF Q

where n is an integer value in the range 0 to 262143, and W, R, F, and Q are simple command characters corresponding to random write, W, random read, R, random write with zero fill, F, and quit processing, Q. If you enter a W or F command, the RANDOM program issues the prompt:

```
type data:
```

You then respond by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If you enter an F command, the RANDOM program fills previously unallocated data blocks with zeros before writing record n. If you enter the R command, RANDOM reads record number n and displays the string value at the console. If you enter the Q command, the X.DAT file is closed, and the program returns to the console command processor. In the interest of brevity, the only error message is:

```
error, try again
```

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label ready where the individual commands are interpreted. The program uses the default file control block at 005CH and the default buffer at 0080H in all disk operations. The utility subroutines that follow contain the principal input line processor, called readc. This particular program shows the elements of random access processing and can be used as the basis for further program development.

```
                        ;*****************************************************
                        ;*                                                   *
                        ;* sample random access program for cp/m 3           *
                        ;*                                                   *
                        ;*****************************************************
0100                            org     100h    ;base of tpa
                        ;
0000 =                  reboot  equ     0000h   ;system reboot
0005 =                  bdos    equ     0005h   ;bdos entry point
                        ;
0001 =                  coninp  equ     1       ;console input function
0002 =                  conout  equ     2       ;console output function
0009 =                  pstring equ     9       ;print string until '$'
000A =                  rstring equ     10      ;read console buffer
000C =                  version equ     12      ;return version number
000F =                  openf   equ     15      ;file open function
0010 =                  closef  equ     16      ;close function
0016 =                  makef   equ     22      ;make file function
0021 =                  readr   equ     33      ;read random
0022 =                  writer  equ     34      ;write random
0028 =                  wrtrzf  equ     40      ;write random zero fill
0098 =                  parsef  equ     152     ;parse function
                        ;
005C =                  fcb     equ     005ch   ;default file control block
007D =                  ranrec  equ     fcb+33  ;random record position
007F =                  ranovf  equ     fcb+35  ;high order (overflow) byte
0080 =                  buff    equ     0080h   ;buffer address
                        ;
000D =                  cr      equ     0dh     ;carriage return
000A =                  lf      equ     0ah     ;line feed
                        ;
```

```
                ;*************************************************
                ;*                                               *
                ;* load SP, set-up file for random access        *
                ;*                                               *
                ;*************************************************
0100 313703             lxi     sp,stack
                ;
                ;       version 3.1?
0103 0E0C               mvi     c,version
0105 CD0500             call    bdos
0108 FE20               cpi     31h     ;version 3.1 or better?
010A D21601             jnc     versok
                ;       bad version, message and go back
010D 118102             lxi     d,badver
0110 CD3102             call    print
0113 C30000             jmp     reboot
                ;
                versok:
                ;       correct version for random access
0116 0E0F               mvi     c,openf ;open default fcb
0118 3A5D00     rdname: lda     fcb+1
011B FE20               cpi     ' '
011D C22C01             jnz     opfile
0120 11E002             lxi     d,entmsg
0123 CD3102             call    print
0126 CD2002             call    parse
0129 C31B01             jmp     rdname
012C 115C00     opfile: lxi     d,fcb
012F CD0500             call    bdos
0132 3C                 inr     a       ;err 255 becomes zero
0133 C24B01             jnz     ready
                ;
                ;       cannot open file, so create it
0136 0E16               mvi     c,makef
0138 115C00             lxi     d,fcb
013B CD0500             call    bdos
013E 3C                 inr     a       ;err 255 becomes zero
013F C24B01             jnz     ready
                ;
                ;       cannot create file, directory full
0142 11A002             lxi     d,nospace
0145 CD3102             call    print
0148 C30000             jmp     reboot  ;back to ccp
                ;
```

```
                ;*************************************************
                ;*                                               *
                ;*  loop back to "ready" after each command      *
                ;*                                               *
                ;*************************************************
                ;
                ready:
                ;       file is ready for processing
                ;
014B CD3C02             call    readcom ;read next command
014E 227D00             shld    ranrec  ;store input record#
0151 217F00             lxi     h,ranovf
0154 71                 mov     m,c     ;set ranrec high byte
0155 FE51               cpi     'Q'     ;quit?
0157 C26901             jnz     notq
                ;
                ;       quit processing, close file
015A 0E10               mvi     c,closef
015C 115C00             lxi     d,fcb
015F CD0500             call    bdos
0162 3C                 inr     a       ;err 255 becomes 0
0163 CAFF01             jz      error   ;error message, retry
0166 C30000             jmp     reboot  ;back to ccp
                ;
                ;*************************************************
                ;*                                               *
                ;* end of quit command, process write            *
                ;*                                               *
                ;*************************************************
                notq:
                ;       not the quit command, random write?
0169 FE57               cpi     'W'
016B C29C01             jnz     notw
                ;
                ;       this is a random write, fill buffer until cr
016E 11B302             lxi     d,datmsg
0171 CD3102             call    print   ;data prompt
0174 0E7F               mvi     c,127   ;up to 127 characters
0176 218000             lxi     h,buff  ;destination
                rloop:  ;read next character to buff
0179 C5                 push    b       ;save counter
017A E5                 push    h       ;next destination
017B CD0802             call    getchr  ;character to a
017E E1                 pop     h       ;restore counter
017F C1                 pop     b       ;restore next to fill
0180 FE0D               cpi     cr      ;end of line?
0182 CA8B01             jz      erloop
```

```
                ;       not end, store character
0185 77                 mov     m,a
0186 23                 inx     h       ;next to fill
0187 0D                 dcr     c       ;counter goes down
0188 C27901             jnz     rloop   ;end of buffer?
                erloop:
                ;       end of read loop, store 00
018B 3600               mvi     m,0
                ;
                ;       write the record to selected record number
018D 0E22               mvi     c,writer
018F 115C00             lxi     d,fcb
0192 CD0500             call    bdos
0195 B7                 ora     a       ;error code zero?
0196 C2FF01             jnz     error   ;message if not
0199 C34B01             jmp     ready   ;for another record
                ;
                ;
                ;******************************************************
                ;*                                                    *
                ;* end of write command, process write random zero fill *
                ;*                                                    *
                ;******************************************************
                notw:
                ;       not the quit command, random write zero fill?
019C FE46               cpi     'F'
019E C2CF01             jnz     notf
                ;
                ;       this is a random write, fill buffer until cr
01A1 11B302             lxi     d,datmsg
01A4 CD3102             call    print   ;data prompt
01A7 0E7F               mvi     c,127   ;up to 127 characters
01A9 218000             lxi     h,buff  ;destination
                rloop1: ;read next character to buff
01AC C5                 push    b       ;save counter
01AD E5                 push    h       ;next destination
01AE CD0802             call    getchr  ;character to a
01B1 E1                 pop     h       ;restore counter
01B2 C1                 pop     b       ;restore next to fill
01B3 FE0D               cpi     cr      ;end of line?
01B5 CABE01             jz      erloop1
                ;       not end, store character
01B8 77                 mov     m,a
01B9 23                 inx     h       ;next to fill
01BA 0D                 dcr     c       ;counter goes down
01BB C2AC01             jnz     rloop1  ;end of buffer?
```

```
                erloop1:
                ;       end of read loop, store 00
01BE 3600               mvi     m,0
                ;
                ;       write the record to selected record number
01C0 0E28               mvi     c,wrtrzf
01C2 115C00             lxi     d,fcb
01C5 CD0500             call    bdos
01C8 B7                 ora     a       ;error code zero?
01C9 C2FF01             jnz     error   ;message if not
01CC C34B01             jmp     ready   ;for another record
                ;
                ;************************************************
                ;*                                              *
                ;* end of write commands, process read          *
                ;*                                              *
                ;************************************************
                notf:
                ;       not a write command, read record?
01CF FE52               cpi     'R'
01D1 C2FF01             jnz     error   ;skip if not
                ;
                ;       read random record
01D4 0E21               mvi     c,readr
01D6 115C00             lxi     d,fcb
01D9 CD0500             call    bdos
01DC B7                 ora     a       ;return code 00?
01DD C2FF01             jnz     error
                ;
                ;       read was successful, write to console
01E0 CD1502             call    crlf    ;new line
01E3 0E80               mvi     c,128   ;max 128 characters
01E5 218000             lxi     h,buff  ;next to get
                wloop:
01E8 7E                 mov     a,m     ;next character
01E9 23                 inx     h       ;next to get
01EA E67F               ani     7fh     ;mask parity
01EC CA4B01             jz      ready   ;for another command if 00
01EF C5                 push    b       ;save counter
01F0 E5                 push    h       ;save next to get
01F1 FE20               cpi     ' '     ;graphic?
01F3 D40E02             cnc     putchr  ;skip output if not
01F6 E1                 pop     h
01F7 C1                 pop     b
01F8 0D                 dcr     c       ;count=count-1
01F9 C2E801             jnz     wloop
01FC C34B01             jmp     ready
```

```
                ;
                ;***************************************************
                ;*                                                 *
                ;* end of read command, all errors end-up here     *
                ;*                                                 *
                ;***************************************************
                ;
                error:
01FF 11BF02             lxi     d,errmsg
0202 CD3102             call    print
0205 C34B01             jmp     ready
                ;
                ;***************************************************
                ;*                                                 *
                ;* utility subroutines for console i/o             *
                ;*                                                 *
                ;***************************************************
                getchr:
                        ;read next console character to a
0208 0E01               mvi     c,coninp
020A CD0500             call    bdos
020D C9                 ret
                ;
                putchr:
                        ;write character from a to console
020E 0E02               mvi     c,conout
0210 5F                 mov     e,a     ;character to send
0211 CD0500             call    bdos    ;send character
0214 C9                 ret
                ;
                crlf:
                        ;send carriage return line feed
0215 3E0D               mvi     a,cr    ;carriage return
0217 CD0E02             call    putchr
021A 3E0A               mvi     a,lf    ;line feed
021C CD0E02             call    putchr
021F C9                 ret
                ;
                parse:
                        ;read and parse filespec
0220 11F102             lxi     d,conbuf
0223 0E0A               mvi     c,rstring
0225 CD0500             call    bdos
0228 111303             lxi     d,pfncb
022B 0E98               mvi     c,parsef
022D CD0500             call    bdos
0230 C9                 ret
                ;
```

```
                print:
                        ;print the buffer addressed by de until $
0231 D5                 push    d
0232 CD1502             call    crlf
0235 D1                 pop     d       ;new line
0236 0E09               mvi     c,pstring
0238 CD0500             call    bdos    ;print the string
023B C9                 ret
                ;
                readcom:
                        ;read the next command line to the conbuf
023C 11D102             lxi     d,prompt
023F CD3102             call    print   ;command?
0242 0E0A               mvi     c,rstring
0244 11F102             lxi     d,conbuf
0247 CD0500             call    bdos    ;read command line
                ;       command line is present, scan it
024A 0E00               mvi     c,0     ;start with 00
024C 210000             lxi     h,0     ;           0000
024F 11F302             lxi     d,conlin;command line
0252 1A         readc:  ldax    d       ;next command character
0253 13                 inx     d       ;to next command position
0254 B7                 ora     a       ;cannot be end of command
0255 C8                 rz
                ;       not zero, numeric?
0256 D630               sui     '0'
0258 FE0A               cpi     10      ;carry if numeric
025A D27902             jnc     endrd
```

```
                ;       add-in next digit
025D F5                 push    psw
025E 79                 mov     a,c     ;value in ahl
025F 29                 dad     h
0260 8F                 adc     a       ;*2
0261 F5                 push    a       ;save value * 2
0262 E5                 push    h
0263 29                 dad     h       ;*4
0264 8F                 adc     a
0265 29                 dad     h       ;*8
0266 8F                 adc     a
0267 C1                 pop     b       ;*2 + *8 = *10
0268 09                 dad     b
0269 C1                 pop     b
026A 88                 adc     b
026B C1                 pop     b       ;+digit
026C 48                 mov     c,b
026D 0600               mvi     b,0
026F 09                 dad     b
0270 CE00               aci     0
0272 4F                 mov     c,a
0273 D25202             jnc     readc
0276 C33C02             jmp     readcom
                endrd:
                ;       end of read, restore value in a
0279 C630               adi     '0'     ;command
027B FE61               cpi     'a'     ;translate case?
027D D8                 rc
                ;       lower case, mask lower case bits
027E E65F               ani     101$1111b
0280 C9                 ret             ;return with value in chl
                ;
                ;*************************************************
                ;*                                               *
                ;* string data area for console messages         *
                ;*                                               *
                ;*************************************************
                badver:
0281 736F727279         db      'sorry, you need cp/m version 3$'
                nospace:
02A0 6E6F206469         db      'no directory space$'
                datmsg:
02B3 7479706520         db      'type data: $'
                errmsg:
02BF 6572726F72         db      'error, try again.$'
                prompt:
02D1 6E65787420         db      'next command? $'
                entmsg:
02E0 656E746572         db      'enter filename: $'
                ;
```

```
                ;*************************************************
                ;*                                               *
                ;* fixed and variable data area                  *
                ;*                                               *
                ;*************************************************
02F1 21         conbuf: db      conlen  ;length of console buffer
02F2            consiz: ds      1       ;resulting size after read
02F3            conlin: ds      32      ;length 32 buffer
0021 =          conlen  equ     $-consiz
                ;
                pfncb:
0313 F302               dw      conlin
0315 5C00               dw      fcb
                ;
0317                    ds      32      ;16 level stack
                stack:
0337                    end
```

You could make the following major improvements to this program to enhance its operation. With some work, this program could evolve into a simple data base management system. You could, for example, assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. You could develop a program called GETKEY that first reads a sequential file and extracts a specific field defined by the operator. For example, the command

```
GETKEY NAMES.DAT  LASTNAME 10 20
```

would cause GETKEY to read the data base file NAMES.DAT and extract the "LASTNAME" field from each record, starting at position 10 and ending at character 20. GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list and writes a new file, called LASTNAME.KEY. This list, sometimes called an inverted index, is an alphabetical list of LASTNAME fields with their corresponding record numbers.

You could rename the program shown above to QUERY, and modify it so that it reads a sorted key file into memory. The command line might appear as

```
QUERY NAMES.DAT LASTNAME.KEY
```

Instead of reading a number, the QUERY program reads an alphanumeric string which is a particular key to find in the NAMES.DAT data base. Because the LAST-NAME.KEY list is sorted, you can find a particular entry quickly by performing a binary search, similar to looking up a name in the telephone directory. Start at both ends of the list and examine the entry halfway in between and, if not matched, split either the upper half or the lower half for the next search. You will quickly reach the item you are looking for, in log2(n) steps, where you will find the corresponding record number. Fetch and display this record at the console as the program illustrates.

At this point, you are just getting started. With a little more work, you can allow a fixed grouping size, which differs from the 128-byte record shown above. You can accomplish this by keeping track of the record number as well as the byte offset within the record. Knowing the group size, you can randomly access the record containing the proper group, offset to the beginning of the group within the record, and read sequentially until the group size has been exhausted.

Finally, you can improve QUERY considerably by allowing Boolean expressions that compute the set of records that satisfy several relationships, such as a LAST-NAME between HARDY and LAUREL and an AGE less than 45. Display all the records that fit this description. Finally, if your lists are getting too big to fit into memory, randomly access your key files from the disk as well.

## 4.4 Construction of an RSX Program

This section describes the standard prefix of a Resident System Extension (RSX) and illustrates the construction of an RSX with an example. (See Section 1.6.4 for a discussion of how RSXs operate under CP/M 3.) RSX programs are usually written in assembler, but you can use other languages if the interface between the language and the calling conventions of the BDOS are set up properly.

### 4.4.1 The RSX Prefix

The first 27 bytes of an RSX program contain a standard data structure called the RSX prefix. The RSX prefix has the following format:

```
serial:
        db      0,0,0,0,0,0
start:
        jmp     ftest                   ; start of program
next:
        db      0c3h                    ; jump instruction to
        dw      0                       ; next module in line
prev:
        dw      0                       ; previous module
remove:
        db      0ffh                    ; remove flag
nonbank:
        db      0                       ; nonbank flag
name:
        db      '12345678'              ; any 8-character name
loader:
        db      0                       ; loader flag
        db      0,0                     ; reserved area
```

The only fields of the RSX prefix that you must initialize are the remove: flag, the nonbank: flag, and the name: of the RSX.

For compatibility with previous releases of CP/M, the serial: field of the prefix is set to the serial number of the operating system by the LOADER module when the RSX is loaded into memory. Thus, the address in location 6 locates the byte following the serial number of the operating system with or without RSXs in memory.

The start: field contains a jump instruction to the beginning of the RSX code where the RSX tests to see if this BDOS function call is to be intercepted or passed on to the next module in line.

The next: field contains a jump instruction to the next module in the chain or the LOADER module if the RSX is the oldest one in memory. The RSX program must make its own BDOS function calls by calling the next: entry point.

The prev: field contains the address of the preceding RSX in memory or location 5 if the RSX is the first RSX in the chain.

The remove: field controls whether the RSX is removed from memory by the next call to the LOADER module via BDOS function 59. If the remove: flag is 0FFH, the LOADER removes the RSX from memory. Note that the CCP always calls the LOADER module during a warm start operation. An RSX that remains in memory past warm start because its remove: flag is zero, must set the flag at its termination to ensure its removal from memory at the following warm start.

The nonbank: field controls when the RSX is loaded. If the field is 0FFH, the LOADER only loads the module into memory on nonbanked CP/M 3 systems. Otherwise, the RSX is loaded into memory under both banked and nonbanked versions of CP/M 3.

The loader: flag identifies the LOADER RSX. When the LOADER module loads an RSX into memory, it sets this prefix flag of the loaded RSX to zero. However, the loader: flag in the LOADER's prefix contains 0FFH. Thus, this flag identifies the last RSX in the chain, which is always the LOADER.

### 4.4.2 Example of RSX Use

These two sample programs illustrate the use of an RSX program. The first program, CALLVERS, prints a message to the console and then makes a BDOS Function 12 call to obtain the CP/M 3 version number. CALLVERS repeats this sequence five times before terminating. The second program, ECHOVERS, is an RSX that intercepts the BDOS Function 12 call made by CALLVERS, prints a second message, and returns the version 0031H to CALLVERS. Although this example is simple, it illustrates BDOS function interception, stack swapping, and BDOS function calls within an RSX.

```
                 ; CALLVERS program
0005 =           bdos    equ     5               ; entry point for BDOS
0009 =           prtstr  equ     9               ; print string function
000C =           vers    equ     12              ; set version function
000D =           cr      equ     0dh             ; carriage return
000A =           lf      equ     0ah             ; line feed

0100                     org     100h
0100 1605                mvi     d,5             ; Perform 5 times
0102 D5          loop:   push    d               ; save counter
0103 0E09                mvi     c,prtstr
0105 111E01              lxi     d,call$msg      ; print call message
0108 CD0500              call    bdos
010B 0E0C                mvi     c,vers
010D CD0500              call    bdos            ; try to set version #
                                                 ; CALLVERS will intercept
0110 7D                  mov     a,l
0111 323401              sta     curvers
0114 D1                  pop     d
0115 15                  dcr     d               ; decrement counter
0116 C20201              jnz     loop
0119 0E00                mvi     c,0
011B C30500              jmp     bdos
                 call$msg:
011E 0D0A2A2A2A          db      cr,lf,'**** CALLVERS **** $'
0134 00          curvers db      0
0135                     end

                 ; ECHOVERS RSX

0009 =           pstring equ     9               ; string print function
000D =           cr      equ     0dh
000A =           lf      equ     0ah
                 ;
                 ;               RSX PREFIX STRUCTURE
                 ;
0000 0000000000          db      0,0,0,0,0,0     ; room for serial number
0006 C31B00              jmp     ftest           ; begin of program
0009 C3          next:   db      0c3H            ; jump
000A 0000                dw      0               ; next module in line
000C 0000        prev:   dw      0               ; previous module
000E FF          remov:  db      0ffh            ; remove flag set
000F 00          nonbnk: db      0
0010 4543484F56          db      'ECHOVERS'
0018 000000              db      0,0,0
```

```
                ftest:                          ; is this function 12?
001B 79                 mov     a,c
001C FE0C               cpi     12
001E CA2400             jz      begin           ; yes - intercept
0021 C30900             jmp     next            ; some other function
                begin:
0024 210000             lxi     h,0
0027 39                 dad     sp              ; save stack
0028 225400             shld    ret$stack
002B 317600             lxi     sp,loc$stack

002E 0E09               mvi     c,pstring
0030 113E00             lxi     d,test$msg      ; print message
0033 CD0900             call    next            ; call BDOS

0036 2A5400             lhld    ret$stack       ; restore user stack
0039 F9                 sphl
003A 213100             lxi     h,0031h         ; return version number
003D C9                 ret

                test$msg:
003E 0D0A2A2A2A         db      cr,lf,'**** ECHOVERS ****$'
                ret$stack:
0054 0000               dw      0
0056                    ds      32              ; 16 level stack
                loc$stack:
0076                    end
```

You can prepare the above programs for execution as follows:

1. Assemble the CALLVERS program using MAC as follows:
   `MAC CALLVERS`
2. Generate a COM file for CALLVERS with HEXCOM:
   `HEXCOM CALLVERS`
3. Assemble the RSX program ECHOVERS using RMAC:
   `RMAC ECHOVERS`
4. Generate a PRL file using the LINK command:
   `LINK ECHOVERS [OP]`
5. Rename the PRL file to an RSX file:
   `RENAME ECHOVERS.RSX=ECHOVERS.PRL`
6. Generate a COM file with an attached RSX using the GENCOM command:
   `GENCOM CALLVERS ECHOVERS`
7. Run the CALLVERS.COM module:
   `CALLVERS`
   The message
   `**** CALLVERS ****`
   followed by the message
   `**** ECHOVERS ****`
   appears on the screen five times if the RSX program works.

*End of Section 4*

# Appendix A
# System Control Block

The System Control Block (SCB) is a CP/M 3 data structure located in the BDOS. CP/M 3 uses this region primarily for communication between the BDOS and the BIOS. However, it is also available for communication between application programs, RSXs, and the BDOS. Note that programs that access the System Control Block are not version independent. They can run only on CP/M 3.

The following list describes the fields of the SCB that are available for access by application programs and RSXs. The location of each field is described as the offset from the start address of the SCB (see BDOS Function 49). The RW/RO column indicates if the SCB field is Read-Write or Read-Only.

**Table A-1. SCB Fields and Definitions**

| *Offset* | *RW/RO* | *Definition* |
|---|---|---|
| 00 — 04 | RO | Reserved for system use. |
| 05 | RO | BDOS Version Number. |
| 06 — 09 | RW | Reserved for user use. Use these four bytes for your own flags or data. |
| 0A — 0F | RO | Reserved for system use. |
| 10 — 11 | RW | Program Error Return Code. This 2-byte field can be used by a program to pass an error code or value to a chained program. CP/M 3's conditional command facility also uses this field to determine if a program executes successfully. The BDOS Function 108 (Get/Set Program Return Code) is used to get/set this value. |
| 12 — 19 | RO | Reserved for system use. |

**Table A-1. (continued)**

| *Offset* | *RW/RO* | *Definition* |
|---|---|---|
| 1A | RW | Console Width. This byte contains the number of columns, characters per line, on your console relative to zero. Most systems default this value to 79. You can set this default value by using the GENCPM or the DEVICE utility. The console width value is used by the banked version of CP/M 3 in BDOS function 10, CP/M 3's console editing input function. Note that typing a character into the last position of the screen, as specified by the Console Width field, must not cause the terminal to advance to the next line. |
| 1B | RO | Console Column Position. This byte contains the current console column position. |
| 1C | RW | Console Page Length. This byte contains the page length, lines per page, of your console. Most systems default this value to 24 lines per page. This default value may be changed by using the GENCPM or the DEVICE utility (see the *CP/M Plus (CP/M Version 3) Operating System User's Guide*). |
| 1D — 21 | RO | Reserved for system use. |
| 22 — 2B | RW | Redirection flags for each of the five logical character devices. If your system's BIOS supports assignment of logical devices to physical devices, you can direct each of the five logical character devices to any combination of up to 12 physical devices. The 16-bit word for each device represents the following:<br>Each bit represents a physical device where bit 15 corresponds to device zero and bit 4 corresponds to device 11. Bits zero through 3 are reserved for system use.<br>You can redirect the input and output logical devices with the DEVICE command (see *CP/M Plus (CP/M Version 3) Operating System User's Guide*). |

**Table A-1. (continued)**

| *Offset* | *RW/RO* | *Definition* |
|---|---|---|
| 22 — 23 | RW | CONIN Redirection Flag. |
| 24 — 25 | RW | CONOUT Redirection Flag. |
| 26 — 27 | RW | AUXIN Redirection Flag. |
| 28 — 29 | RW | AUXOUT Redirection Flag. |
| 2A — 2B | RW | LSTOUT Redirection Flag. |
| 2C | RW | Page Mode. If this byte is set to zero, some CP/M 3 utilities and CCP built-in commands display one page of data at a time; you display the next page by pressing any key. If this byte is not set to zero, the system displays data on the screen without stopping. To stop and start the display, you can press CTRL-S and CTRL-Q, respectively. |
| 2D | RO | Reserved for system use. |
| 2E | RW | Determines if CTRL-H is interpreted as a rub/del character. If this byte is set to 0, then CTRL-H is a backspace character (moves back and deletes). If this byte is set to 0FFH, then CTRL-H is a rub/del character, echoes the deleted character. |
| 2F | RW | Determines if rub/del is interpreted as CTRL-H character. If this byte is set to 0, then rub/del echoes the deleted character. If this byte is set to 0FF, then rub/del is interpreted as a CTRL-H character (moves back and deletes). |
| 30 — 32 | RO | Reserved for system use. |
| 33 — 34 | RW | Console Mode. This is a 16-bit system parameter that determines the action of certain BDOS Console I/O functions. (See Section 2.2.1 and BDOS Function 109, Get/Set Console Mode, for a thorough explanation of Console Mode.) |

**Table A-1. (continued)**

| *Offset* | *RW/RO* | *Definition* |
|---|---|---|
| 35 — 36 | RO | Reserved for system use. |
| 37 | RW | Output delimiter character. The default output delimiter character is $, but you can change this value by using the BDOS Function 110, Get/Set Output Delimiter. |
| 38 | RW | List Output Flag. If this byte is set to 0, console output is not echoed to the list device. If this byte is set to 1 console output is echoed to the list device. |
| 39 — 3B | RO | Reserved for system use. |
| 3C — 3D | RO | Current DMA Address. This address can be set by BDOS Function 26 (Set DMA Address). The CCP initializes this value to 0080H. BDOS Function 13, Reset Disk System, also sets the DMA address to 0080H. |
| 3E | RO | Current Disk. This byte contains the currently selected default disk number. This value ranges from 0-15 corresponding to drives A-P, respectively. BDOS Function 25, Return Current Disk, can be used to determine the current disk value. |
| 3F — 43 | RO | Reserved for system use. |
| 44 | RO | Current User Number. This byte contains the current user number. This value ranges from 0-15. BDOS Function 32, Set/Get User Code, can change or interrogate the currently active user number. |
| 45 — 49 | RO | Reserved for system use. |
| 4A | RW | BDOS Multi-Sector Count. This field is set by BDOS Function 44, Set Multi-Sector Count. |

**Table A-1. (continued)**

| *Offset* | *RW/RO* | *Definition* |
|---|---|---|
| 4B | RW | BDOS Error Mode. This field is set by BDOS Function 45, Set BDOS Error Mode.<br><br>If this byte is set to 0FFH, the system returns to the current program without displaying any error messages. If it is set to 0FEH, the system displays error messages before returning to the current program. Otherwise, the system terminates the program and displays error messages. See description of BDOS Function 45, Set BDOS Error Mode, for discussion of the different error modes. |
| 4C — 4F | RW | Drive Search Chain. The first byte contains the drive number of the first drive in the chain, the second byte contains the drive number of the second drive in the chain, and so on, for up to four bytes. If less than four drives are to be searched, the next byte is set to 0FFH to signal the end of the search chain. The drive values range from 0-16, where 0 corresponds to the default drive, while 1-16 corresponds to drives A-P, respectively. The drive search chain can be displayed or set by using the SETDEF utility (see *CP/M Plus (Version 3) Operating System User's Guide*). |
| 50 | RW | Temporary File Drive. This byte contains the drive number of the temporary file drive. The drive number ranges from 0-16, where 0 corresponds to the default drive, while 1-16 corresponds to drives A-P, respectively. |
| 51 | RO | Error drive. This byte contains the drive number of the selected drive when the last physical or extended error occurred. |
| 52 — 56 | RO | Reserved for system use. |

**Table A-1. (continued)**

| *Offset* | *RW/RO* | *Definition* |
|---|---|---|
| 57 | RO | BDOS Flags. Bit 7 applies to banked systems only. If bit 7 is set, then the system displays expanded error messages. The second error line displays the function number and FCB information. (See Section 2.3.13).<br>Bit 6 applies only to nonbanked systems. If bit 6 is set, it indicates that GENCPM has specified single allocation vectors for the system. Otherwise, double allocation vectors have been defined for the system. Function 98, Free Blocks, returns temporarily allocated blocks to free space only if bit 6 is reset. |
| 58 — 59 | RW | Date in days in binary since 1 Jan 78. |
| 5A | RW | Hour in BCD (2-digit Binary Coded Decimal). |
| 5B | RW | Minutes in BCD. |
| 5C | RW | Seconds in BCD. |
| 5D — 5E | RO | Common Memory Base Address. This value is zero for nonbanked systems and nonzero for banked systems. |
| 5F — 63 | RO | Reserved for system use. |

*End of Appendix A*

# Appendix B
# PRL File Generation

## B.1 PRL Format

A Page Relocatable Program has an origin offset of 100H bytes that is stored on disk as a file of type PRL. The format is shown in Table B-1.

**Table B-1. PRL File Format**

| *Address* | *Contents* |
|---|---|
| 0001-0002H | Program size |
| 0004-0005H | Minimum buffer requirements (additional memory) |
| 0006-00FFH | Currently unused, reserved for future allocation |
| 0100 + Program size = Start of bit map | |

The bit map is a string of bits identifying those bytes in the source code that require relocation. There is one byte in the bit map for every 8 bytes of source code. The most significant bit, bit 7, of the first byte of the bit map indicates whether or not the first byte of the source code requires relocation. If the bit is on, it indicates that relocation is required. The next bit, bit 6, of the first byte corresponds to the second byte of the source code, and so forth.

## B.2 Generating a PRL

The preferred technique for generating a PRL file is to use the CP/M LINK-80™, which can generate a PRL file from a REL relocatable object file. This technique is described in the *Programmer's Utilities Guide for The CP/M Family of Operating Systems*. A sample link command is shown below.

```
A>link dump[op]
```

*End of Appendix B*

# Appendix C
# SPR Generation

System Page Relocatable, SPR, files are similar in format to PRL files except that SPR files have an origin offset of 0000H (see Appendix B). SPR Files are provided as part of the standard CP/M 3 System: the resident and banked portions of the banked BDOS, named RESBDOS3.SPR and BNKBDOS3.SPR, and the nonbanked BDOS, named BDOS3.SPR. The customized BIOS must also be generated in SPR format before GENCPM can create a CP/M 3 system. The BIOS SPR file is named BNKBIOS3.SPR for banked systems and BIOS3.SPR for nonbanked systems. A detailed discussion of the generation of BIOS3.SPR or BNKBIOS3.SPR is provided in the *CP/M Plus (CP/M Version 3) Operating System System Guide.*

The method of generating an SPR is analogous to that of generating a Page Relocatable Program (described in Appendix B) with the following exceptions:

- If LINK-80 is used, the output file of type SPR is specified with the [os] or [b] option. The [b] option is used when linking BNKBIOS3.SPR.
- The code in the SPR is ORGed at 000H rather than 100H.

*End of Appendix C*

# Appendix D
# ASCII and Hexadecimal Conversions

This appendix contains tables of the ASCII symbols, including their binary, decimal, and hexadecimal conversions.

**Table D-1. ASCII Symbols**

| *Symbol* | *Meaning* | *Symbol* | *Meaning* |
|---|---|---|---|
| ACK | acknowledge | FS | file separator |
| BEL | bell | GS | group separator |
| BS | backspace | HT | horizontal tabulation |
| CAN | cancel | LF | line-feed |
| CR | carriage return | NAK | negative acknowledge |
| DC | device control | NUL | null |
| DEL | delete | RS | record separator |
| DLE | data link escape | SI | shift in |
| EM | end of medium | SO | shift out |
| ENQ | enquiry | SOH | start of heading |
| EOT | end of transmission | SP | space |
| ESC | escape | STX | start of text |
| ETB | end of transmission | SUB | substitute |
| ETX | end of text | SYN | synchronous idle |
| FF | form-feed | US | unit separator |
| | | VT | vertical tabulation |

**Table D-2. ASCII Conversion Table**

| *Binary* | *Decimal* | *Hexadecimal* | *ASCII* |
|---|---|---|---|
| 0000000 | 000 | 00 | NUL |
| 0000001 | 001 | 01 | SOH (CTRL-A) |
| 0000010 | 002 | 02 | STX (CTRL-B) |
| 0000011 | 003 | 03 | ETX (CTRL-C) |
| 0000100 | 004 | 04 | EOT (CTRL-D) |
| 0000101 | 005 | 05 | ENQ (CTRL-E) |
| 0000110 | 006 | 06 | ACK (CTRL-F) |
| 0000111 | 007 | 07 | BEL (CTRL-G) |
| 0001000 | 008 | 08 | BS (CTRL-H) |
| 0001001 | 009 | 09 | HT (CTRL-I) |
| 0001010 | 010 | 0A | LF (CTRL-J) |
| 0001011 | 011 | 0B | VT (CTRL-K) |
| 0001100 | 012 | 0C | FF (CTRL-L) |
| 0001101 | 013 | 0D | CR (CTRL-M) |
| 0001110 | 014 | 0E | SO (CTRL-N) |
| 0001111 | 015 | 0F | SI (CTRL-O) |
| 0010000 | 016 | 10 | DLE (CTRL-P) |
| 0010001 | 017 | 11 | DC1 (CTRL-Q) |
| 0010010 | 018 | 12 | DC2 (CTRL-R) |
| 0010011 | 019 | 13 | DC3 (CTRL-S) |
| 0010100 | 020 | 14 | DC4 (CTRL-T) |
| 0010101 | 021 | 15 | NAK (CTRL-U) |
| 0010110 | 022 | 16 | SYN (CTRL-V) |
| 0010111 | 023 | 17 | ETB (CTRL-W) |
| 0011000 | 024 | 18 | CAN (CTRL-X) |
| 0011001 | 025 | 19 | EM (CTRL-Y) |
| 0011010 | 026 | 1A | SUB (CTRL-Z) |
| 0011011 | 027 | 1B | ESC (CTRL-[) |
| 0011100 | 028 | 1C | FS (CTRL-\) |
| 0011101 | 029 | 1D | GS (CTRL-]) |
| 0011110 | 030 | 1E | RS (CTRL-^) |
| 0011111 | 031 | 1F | US (CTRL-_) |
| 0100000 | 032 | 20 | (SPACE) |
| 0100001 | 033 | 21 | ! |
| 0100010 | 034 | 22 | " |
| 0100011 | 035 | 23 | # |
| 0100100 | 036 | 24 | $ |

Table D-2. (continued)

| *Binary* | *Decimal* | *Hexadecimal* | *ASCII* |
|---|---|---|---|
| 0100101 | 037 | 25 | % |
| 0100110 | 038 | 26 | & |
| 0100111 | 039 | 27 | ' |
| 0101000 | 040 | 28 | ( |
| 0101001 | 041 | 29 | ) |
| 0101010 | 042 | 2A | * |
| 0101011 | 043 | 2B | + |
| 0101100 | 044 | 2C | , |
| 0101101 | 045 | 2D | - |
| 0101110 | 046 | 2E | . |
| 0101111 | 047 | 2F | / |
| 0110000 | 048 | 30 | 0 |
| 0110001 | 049 | 31 | 1 |
| 0110010 | 050 | 32 | 2 |
| 0110011 | 051 | 33 | 3 |
| 0110100 | 052 | 34 | 4 |
| 0110101 | 053 | 35 | 5 |
| 0110110 | 054 | 36 | 6 |
| 0110111 | 055 | 37 | 7 |
| 0111000 | 056 | 38 | 8 |
| 0111001 | 057 | 39 | 9 |
| 0111010 | 058 | 3A | : |
| 0111011 | 059 | 3B | ; |
| 0111100 | 060 | 3C | < |
| 0111101 | 061 | 3D | = |
| 0111110 | 062 | 3E | > |
| 0111111 | 063 | 3F | ? |
| 1000000 | 064 | 40 | @ |
| 1000001 | 065 | 41 | A |
| 1000010 | 066 | 42 | B |
| 1000011 | 067 | 43 | C |
| 1000100 | 068 | 44 | D |
| 1000101 | 069 | 45 | E |
| 1000110 | 070 | 46 | F |
| 1000111 | 071 | 47 | G |
| 1001000 | 072 | 48 | H |
| 1001001 | 073 | 49 | I |

**Table D-2. (continued)**

| *Binary* | *Decimal* | *Hexadecimal* | *ASCII* |
|---|---|---|---|
| 1001010 | 074 | 4A | J |
| 1001011 | 075 | 4B | K |
| 1001100 | 076 | 4C | L |
| 1001101 | 077 | 4D | M |
| 1001110 | 078 | 4E | N |
| 1001111 | 079 | 4F | O |
| 1010000 | 080 | 50 | P |
| 1010001 | 081 | 51 | Q |
| 1010010 | 082 | 52 | R |
| 1010011 | 083 | 53 | S |
| 1010100 | 084 | 54 | T |
| 1010101 | 085 | 55 | U |
| 1010110 | 086 | 56 | V |
| 1010111 | 087 | 57 | W |
| 1011000 | 088 | 58 | X |
| 1011001 | 089 | 59 | Y |
| 1011010 | 090 | 5A | Z |
| 1011011 | 091 | 5B | [ |
| 1011100 | 092 | 5C | \ |
| 1011101 | 093 | 5D | ] |
| 1011110 | 094 | 5E | ^ |
| 1011111 | 095 | 5F | < |
| 1100000 | 096 | 60 | ` |
| 1100001 | 097 | 61 | a |
| 1100010 | 098 | 62 | b |
| 1100011 | 099 | 63 | c |
| 1100100 | 100 | 64 | d |
| 1100101 | 101 | 65 | e |
| 1100110 | 102 | 66 | f |

**Table D-2. (continued)**

| *Binary* | *Decimal* | *Hexadecimal* | *ASCII* |
|---|---|---|---|
| 1100111 | 103 | 67 | g |
| 1101000 | 104 | 68 | h |
| 1101001 | 105 | 69 | i |
| 1101010 | 106 | 6A | j |
| 1101011 | 107 | 6B | k |
| 1101100 | 108 | 6C | l |
| 1101101 | 109 | 6D | m |
| 1101110 | 110 | 6E | n |
| 1101111 | 111 | 6F | o |
| 1110000 | 112 | 70 | p |
| 1110001 | 113 | 71 | q |
| 1110010 | 114 | 72 | r |
| 1110011 | 115 | 73 | s |
| 1110100 | 116 | 74 | t |
| 1110101 | 117 | 75 | u |
| 1110110 | 118 | 76 | v |
| 1110111 | 119 | 77 | w |
| 1111000 | 120 | 78 | x |
| 1111001 | 121 | 79 | y |
| 1111010 | 122 | 7A | z |
| 1111011 | 123 | 7B | { |
| 1111100 | 124 | 7C | \| |
| 1111101 | 125 | 7D | } |
| 1111110 | 126 | 7E | ~ |
| 1111111 | 127 | 7F | DEL |

*End of Appendix D*

# Appendix E
# BDOS Function Summary

**Table E-1. BDOS Function Summary**

| *Function* | *Function Name* | *Input Parameters* | *Returned Values* |
|---|---|---|---|
| 0 | System Reset | none | none |
| 1 | Console Input | none | A = char |
| 2 | Console Output | E = char | A = 00H |
| 3 | Auxiliary Input | none | A = char |
| 4 | Auxiliary Output | E = char | A = 00H |
| 5 | List Output | E = char | A = 00H |
| 6 | Direct Console I/O | E = 0FFH/ 0FEH/ 0FDH/ char | A = char/status/ none |
| 7 | Auxiliary Input Status | none | A = 00/0FFH |
| 8 | Auxiliary Output Status | none | A = 00/0FFH |
| 9 | Print String | DE = .String | A = 00H |
| 10 | Read Console Buffer | DE = .Buffer0 | Characters in buffer |
| 11 | Get Console Status | none | A = 00/01 |
| 12 | Return Version Number | none | HL = Version (0031H) |
| 13 | Reset Disk System | none | A = 00H |
| 14 | Select Disk | E = Disk Number | A = Err Flag |
| 15 | Open File | DE = .FCB | A = Dir Code |
| 16 | Close File | DE = .FCB | A = Dir Code |
| 17 | Search for First | DE = .FCB | A = Dir Code |
| 18 | Search for Next | none | A = Dir Code |
| 19 | Delete File | DE = .FCB | A = Dir Code |
| 20 | Read Sequential | DE = .FCB | A = Err Code |
| 21 | Write Sequential | DE = .FCB | A = Err Code |
| 22 | Make File | DE = .FCB | A = Dir Code |
| 23 | Rename File | DE = .FCB | A = Dir Code |
| 24 | Return Login Vector | none | HL = Login Vector |
| 25 | Return Current Disk | none | A = Cur Disk# |

Table E-1. (continued)

| *Function* | *Function Name* | *Input Parameters* | *Returned Values* |
|---|---|---|---|
| 25 | Return Current Disk | none | A = Cur Disk# |
| 26 | Set DMA Address | DE = .DMA | A = 00H |
| 27 | Get Addr(Alloc) | none | HL = .Alloc |
| 28 | Write Protect Disk | none | A = 00H |
| 29 | Get R/O Vector | none | HL = R/O Vector |
| 30 | Set File Attributes | DE = .FCB | A = Dir Code |
| 31 | Get Addr(DPB) | none | HL = .DPB |
| 32 | Set/Get User Code | E = 0FFH/ user number | A = Curr User/ 00H |
| 33 | Read Random | DE = .FCB | A = Err Code |
| 34 | Write Random | DE = .FCB | A = Err Code |
| 35 | Compute File Size | DE = .FCB | r0, r1, r2<br>A = Err Flag |
| 36 | Set Random Record | DE = .FCB | r0, r1, r2 |
| 37 | Reset Drive | DE = Drive Vector | A = 00H |
| 38 | Access Drive | none | A = 00H |
| 39 | Free Drive | none | A = 00H |
| 40 | Write Random with Zero Fill | DE = .FCB | A = Err Code |
| 41 | Test and Write Record | DE = .FCB | A = 0FFH |
| 42 | Lock Record | DE = .FCB | A = 00H |
| 43 | Unlock Record | DE = .FCB | A = 00H |
| 44 | Set Multi-sector Count | E = # Sectors | A = Return Code |
| 45 | Set BDOS Error Mode | E = BDOS Err Mode | A = 00H |
| 46 | Get Disk Free Space | E = Drive number | Number of Free Sectors<br>A = Err Flag |
| 47 | Chain to Program | E = Chain Flag | A = 00H |
| 48 | Flush Buffers | E = Purge Flag | A = Err Flag |
| 49 | Get/Set System Control Block | DE = .SCB PB | A = Returned Byte<br>HL = Returned Word |
| 50 | Direct BIOS Calls | DE = .BIOS PB | BIOS Return |
| 59 | Load Overlay | DE = .FCB | A = Err Code |
| 60 | Call Resident System Extension | DE = .RSX PB | A = Err Code |

**Note:** . indicates the address of

**Table E-1. (continued)**

| *Function* | *Function Name* | *Input Parameters* | *Returned Values* |
|---|---|---|---|
| 98 | Free Blocks | none | A = Err Flag |
| 99 | Truncate File | DE = .FCB | A = Dir Code |
| 100 | Set Directory Label | DE = .FCB | A = Dir Code |
| 101 | Return Directory Label Data | E = Drive | A = Dir label data byte |
| 102 | Read File Date Stamps and Password Mode | DE = .FCB | A = Dir Code |
| 103 | Write File XFCB | DE = .FCB | A = Dir Code |
| 104 | Set Date and Time | DE = .DAT | A = 00H |
| 105 | Get Date and Time | DE = .DAT | Date and Time A = seconds |
| 106 | Set Default Password | DE = .Password | A = 00H |
| 107 | Return Serial Number | DE = .Serial # field | Serial Number |
| 108 | Get/Set Program Return Code | DE = 0FFFFH/ Code | HL Program Ret Code none |
| 109 | Get/Set Console Mode | DE = 0FFFFH/ Mode | HL = Console Mode none |
| 110 | Get/Set Output Delimiter | DE = 0FFFFH/ E = Delimiter | A = Output Delimiter none |
| 111 | Print Block | DE = .CCB | A = 00H |
| 112 | List Block | DE = .CCB | A = 00H |
| 152 | Parse Filename | DE = .PFCB | See definition |

**Note:** . indicates the address of

*End of Appendix E*

# Index

## E



## F

**S**

**T**



**U**



**V**



**W**

## NOTES

# NOTES

# CP/M Plus™

(CP/M® Version 3.0)
Operating System

# System Guide

NOTICE TO USER

From time to time changes are made in the filenames and in the files actually included on the distribution disk. This manual should not be construed as a representation or warranty that such files or facilities exist on the distribution disk or as part of the materials and programs distributed. Most distribution disks include a "README.DOC" file. This file explains variations from the manual which do constitute modification of the manual and the items included therewith. Be sure to read this file before using the software.



The CP/M Plus (CP/M Version 3) Operating System System Guide was prepared using the Digital Research TEX™ Text Formatter and printed in the United States of America.

```
***********************************
*  First Edition:  January 1983   *
*   Second Edition:  July 1983    *
***********************************
```

# Foreword

CP/M® 3, also marketed as CP/M Plus™ , is a single-console operating system for 8-bit machines that use an Intel® 8080, 8085, or Zilog® Z80® CPU. CP/M 3 is upward-compatible with its predecessor, CP/M 2, and offers more features and higher performance than CP/M 2. This manual describes the steps necessary to create or modify a CP/M 3 Basic Input Output System (BIOS) tailored for a specific hardware environment.

The CP/M Plus (CP/M Version 3) Operating System System Guide (hereafter cited as CP/M Plus System Guide) assumes you are familiar with systems programming in 8080 assembly language and that you have access to a CP/M 2 system. It also assumes you understand the target hardware and that you have functioning disk I/O drivers. You should be familiar with the accompanying CP/M Plus (CP/M Version 3) Operating System User's Guide (hereafter cited as CP/M Plus User's Guide) describing the operating system utilities. You should also be familiar with the CP/M Plus (CP/M Version 3) Operating System Programmer's Guide (hereafter cited as CP/M Plus Programmer's Guide), which describes the system calls used by the applications programmer to interface with the operating system. The Programmer's Utilities Guide for the CP/M Family of Operating Systems (hereafter cited as Programmer's Utilities Guide) documents the assembling and debugging utilities.

Section 1 of this manual is an overview of the component modules of the CP/M 3 operating system. Section 2 provides an overview of the functions and data structures necessary to write an interface module between CP/M 3 and specific hardware. Section 3 contains a detailed description of these functions and data structures, followed by instructions to assemble and link the distributed modules with your customized modules. Section 4 describes the modular organization of the sample CP/M 3 BIOS on your distribution diskette. Section 5 documents the procedure to generate and boot your CP/M 3 system. Section 6 is a sample debugging session.

The appendixes contain tables, and sample BIOS modules you can use, or study and modify. Appendix A discusses removable media drives. Appendix B discusses automatic density support. Appendix C describes how CP/M 3 differs from CP/M 2. Appendix D shows the format of the CPM3.SYS file.

Appendixes E through H are listings of the assembled source code for the four hardware-independent modules of the sample BIOS. Appendix E is the kernel module to use when creating a modular BIOS in the form of the distributed sample. Appendix F shows the System Control Block. Appendix G is a table of equates for the baud rate and mode byte for character I/O. Appendix H contains the macro definitions you can use to generate some of the CP/M 3 disk data structures. Appendix I lists the assembled source code for the six BIOS modules that depend on the Altos® 8000-15 Computer System hardware. It also contains a sample Submit file to build a BIOS.

Appendixes J and K are tabular summaries of the public entry points and data items in the modules of the sample BIOS. Finally, Appendix L is a tabular summary of the thirty-three functions of the CP/M 3 BIOS, complete with entry parameters and returned values.

# Table of Contents

# Table of Contents (continued)

# Table of Contents
(continued)

# Appendixes

# Tables, Figures, and Listings

**Tables**

# Tables, Figures, and Listings
(continued)

## Figures



## Listings

# Section 1
# CP/M 3 Operating System Overview

This section is an overview of the CP/M 3 operating system, with a description of the system components and how they relate to each other. The section includes a discussion of memory configurations and supported hardware. The last portion summarizes the creation of a customized version of the CP/M 3 Basic Input Output System (BIOS).

## 1.1 Introduction to CP/M 3

CP/M 3 provides an environment for program development and execution on computer systems that use the Intel 8080, 8085, or Z80 microprocessor chip. CP/M 3 provides rapid access to data and programs through a file structure that supports dynamic allocation of space for sequential and random access files.

CP/M 3 supports a maximum of sixteen logical floppy or hard disks with a storage capacity of up to 512 megabytes each. The maximum file size supported is 32 megabytes. You can configure the number of directory entries and block size to satisfy various user needs.

CP/M 3 is supplied in two versions. One version supports nonbank-switched memory; the second version supports hardware with bank-switched memory capabilities. CP/M 3 supplies additional facilities for the bank-switched system, including extended command line editing, password protection of files, and extended error messages.

The nonbanked system requires 8.5 kilobytes of memory, plus space for your customized BIOS. It can execute in a minimum of 32 kilobytes of memory.

The bank-switched system requires a minimum of two memory banks with 11 kilobytes of memory in Bank 0 and 1.5 kilobytes in common memory, plus space for your customized BIOS. The bank-switched system provides more user memory for application programs.

CP/M 3 resides in the file CPM3.SYS, which is loaded into memory by a system loader during system initialization. The system loader resides on the first two tracks of the system disk. CPM3.SYS contains the distributed BDOS and the customized BIOS.

The CP/M 3 operating system is distributed on two single-density, single-sided, eight-inch floppy disks. Digital Research supplies a sample BIOS that is configured for an Altos 8000-15 microcomputer system with bank-switched memory and two single-density, single-sided, eight-inch floppy disk drives.

## 1.2 CP/M 3 System Components

The CP/M 3 operating system consists of the following three modules: the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the Basic Input Output System (BIOS).

The CCP is a program that provides the basic user interface to the facilities of the operating system. The CCP supplies six built-in commands: DIR, DIRS, ERASE, RENAME, TYPE, and USER. The CCP executes in the Transient Program Area (TPA), the region of memory where all application programs execute. The CCP contains the Program Loader Module, which loads transient (applications) programs from disk into the TPA for execution.

The BDOS is the logical nucleus and file system of CP/M 3. The BDOS provides the interface between the application program and the physical input/output routines of the BIOS.

The BIOS is a hardware-dependent module that interfaces the BDOS to a particular hardware environment. The BIOS performs all physical I/O in the system. The BIOS consists of a number of routines that you must configure to support the specific hardware of the target computer system.

The BDOS and the BIOS modules cooperate to provide the CCP and other transient programs with hardware-independent access to CP/M 3 facilities. Because the BIOS is configured for different hardware environments and the BDOS remains constant, you can transfer programs that run under CP/M 3 unchanged to systems with different hardware configurations.

## 1.3 Communication Between Modules

The BIOS loads the CCP into the TPA at system cold and warm start. The CCP moves the Program Loader Module to the top of the TPA and uses the Program Loader Module to load transient programs.

The BDOS contains a set of functions that the CCP and applications programs call to perform disk and character input and output operations.

The BIOS contains a Jump Table with a set of 33 entry points that the BDOS calls to perform hardware-dependent primitive functions, such as peripheral device I/O. For example, CONIN is an entry point of the BIOS called by the BDOS to read the next console input character.

Similarities exist between the BDOS functions and the BIOS functions, particularly for simple device I/O. For example, when a transient program makes a console output function call to the BDOS, the BDOS makes a console output call to the BIOS. In the case of disk I/O, however, this relationship is more complex. The BDOS might make many BIOS function calls to perform a single BDOS file I/O function. BDOS disk I/O is in terms of 128-byte logical records. BIOS disk I/O is in terms of physical sectors and tracks.

The System Control Block (SCB) is a 100-byte, decimal, CP/M 3 data structure that resides in the BDOS system component. The BDOS and the BIOS communicate through fields in the SCB. The SCB contains BDOS flags and data, CCP flags and data, and other system information, such as console characteristics and the current date and time. You can access some of the System Control Block fields from the BIOS.

Note that the SCB contains critical system parameters which reflect the current state of the operating system. If a program modifies these parameters, the operating system can crash. See Section 3 of this manual, and the description of BDOS Function 49 in the CP/M Plus Programmer's Guide for more information on the System Control Block.

Page Zero is a region of memory that acts as an interface between transient programs and the operating system. Page Zero contains critical system parameters, including the entry to the BDOS and the entry to the BIOS Warm BOOT routine. At system start-up, the BIOS initializes these two entry points in Page Zero. All linkage between transient programs and the BDOS is restricted to the indirect linkage through Page Zero. Figure 1-1 illustrates the general memory organization of CP/M 3.

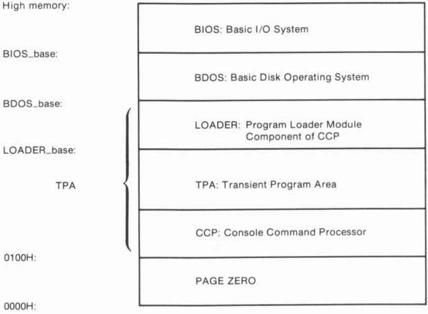


**Figure 1-1. General Memory Organization of CP/M 3**

Note that all memory regions in CP/M 3 are page aligned, which means that they must begin on a page boundary. Because a page is defined as 256 (100H) bytes, a page boundary always begins at a hexadecimal address where the low-order byte of the hex address is zero.

## 1.4 Banked and Nonbanked Systems

CP/M 3 is supplied in two versions: one for hardware that supports banked memory, and the other for hardware with a minimum of 32 kilobytes of memory. The systems are called banked and nonbanked.

Digital Research supplies System Page Relocatable (.SPR) files for both a banked BDOS and a nonbanked BDOS. A sample banked BIOS is supplied for you to use as an example when creating a customized BIOS for your set of hardware components.

The following figure shows the memory organization for a banked system. Bank 0 and common memory are for the operating system. Bank 1 is the Transient Program Area, which contains the Page Zero region of memory. You can use additional banks to enhance operating system performance.

In banked CP/M 3 systems, CPMLDR, the system loader, loads part of the BDOS into common memory and part of the BDOS into Bank 0. CPMLDR loads the BIOS in the same manner.

Figure 1-2 shows the memory organization for the banked version of CP/M 3.

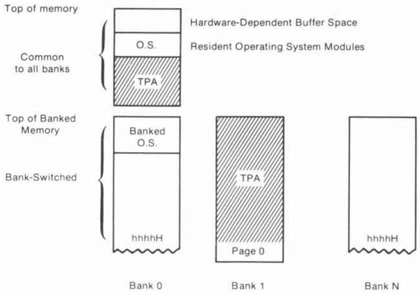


**Figure 1-2. Memory Organization for Banked CP/M 3 System**

In this figure, the top region of memory is called common memory. Common memory is always enabled and addressable. The operating system is divided into two modules: the resident portion, which resides in common memory, and the banked portion, which resides just below common memory in Bank 0.

The shaded areas in Figure 1-2 represent the memory available to transient programs. The clear areas are used by the operating system for disk record buffers and directory hash tables. The clear area in the common region above the operating system represents

space that can be allocated for data buffers by GENCPM, the CP/M 3 system generation utility. The minimum size of the buffer area is determined by the specific hardware requirements of the host microcomputer system.

Bank 0, the system bank, is the bank that is enabled when CP/M 3 is cold started. Bank 1 is the transient program bank.

The transient program bank must be contiguous from location zero to the top of banked memory. Common memory must also be contiguous. The other banks need not begin at location zero or have contiguous memory.

Figure 1-3 shows the CP/M 3 memory organization when the TPA bank, Bank 1, is enabled in a bank-switched system.

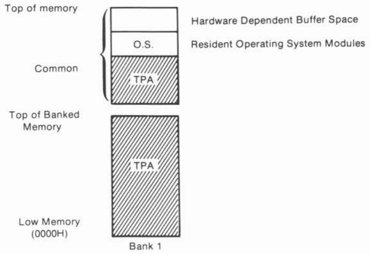


**Figure 1-3. Memory Organization with Bank 1 Enabled in Banked System**

The operating system switches to Bank 0 or other banks when performing operating system functions. In general, any bank switching performed by the operating system is transparent to the calling program.

The memory organization for the nonbanked version of CP/M 3 is much simpler, as shown in Figure 1-4:

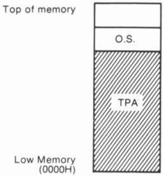


**Figure 1-4. Memory Organization in Nonbanked CP/M 3 System**

In the nonbanked version of CP/M 3, memory consists of a single contiguous region addressable from 0000H up to a maximum of 0FFFFH, or 64K-1. The clear area above the operating system represents space that can be allocated for data buffers and directory hash tables by the CP/M 3 system generation utility, GENCPM, or directly allocated by the BIOS. The minimum size of the buffer area is determined by the specific hardware requirements of the host microcomputer system. Again, the shaded region represents the space available for transient programs.

## 1.5 Memory Requirements

Table 1-1 shows typical sizes of the CP/M 3 operating system components.

**Table 1-1. CP/M 3 Operating System Memory Requirements**

| CP/M 3 Version | Nonbanked | Banked | |
|---|---|---|---|
| | | Common | Bank 0 |
| BDOS | 8.5K | 1.5K | 11K |
| BIOS (values vary) | | | |
| floppy system | 1.5K | .75K | 2K |
| hard system | 2.5K | 1.5K | 3K |

The CP/M 3 banked system requires a minimum of two banks (Bank 0 and Bank 1) and can support up to 16 banks of memory. The size of the common region is often 16K, but can be as small as 4K. Common memory must be large enough to contain the required buffers and the resident (common) portion of the operating system, which means a 1.5K BDOS and the common part of your customized BIOS.

In a banked environment, CP/M 3 maintains a cache of deblocking buffers and directory records using a Least Recently Used (LRU) buffering scheme. The LRU buffer is the first to be reused when the system runs out of buffer space. The BDOS maintains separate buffer pools for directory and data record caching.

The RSX modules shown in Figure 1-5 are Resident System Extensions (RSX) that are loaded directly below the operating system when included in an application or utility program. The Program Loader places the RSX in memory and chains BDOS calls through the RSX entry point in the RSX.

Figure 1-5 shows the memory organization in a typical bank-switched CP/M 3 system.

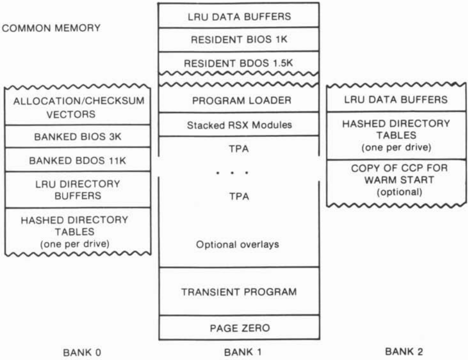


**Figure 1-5. Memory Organization in Banked CP/M 3**

The banked system supports a TPA of 60K or more. The banked portion of the operating system in Bank 0 requires at least 16K of memory.

In the banked system, the BDOS and the BIOS are separated into two parts: a resident portion, and a banked portion. The resident BDOS and BIOS are located in common memory. The banked BDOS and BIOS are located in the operating system bank, referred to as Bank 0 in this manual.

The TPA extends from 100H in Bank 1 up to the bottom of the resident BDOS in common memory. The banked BIOS and BDOS reside in Bank 0 with the directory buffers. Typically, all data buffers reside in common. Data buffers can reside in an alternate bank if the system has a DMA controller capable of transferring arbitrary blocks of data from one bank to another. Hashed directory tables (one per drive) can be placed in any bank except Bank 1 (TPA). Hashed directory tables require 4 bytes per directory entry.

Figure 1-6 shows a typical nonbanked system configuration.

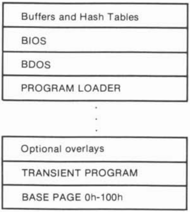


**Figure 1-6. Memory Organization in Nonbanked CP/M 3**

The nonbanked CP/M 3 system requires 8.5K of memory plus space for the BIOS, buffers, and hash tables, allowing a TPA size of up to 52K to 54K, depending on the size of the BIOS and the number of hash tables and buffers you are using.

## 1.6 Disk Organization

Figure 1-7 illustrates the organization of a CP/M 3 system disk.

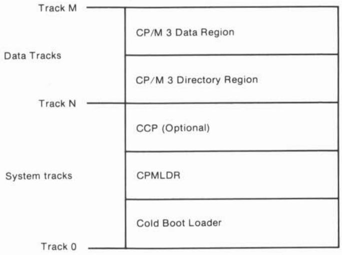


**Figure 1-7. CP/M 3 System Disk Organization**

In Figure 1-7, the first N tracks are the system tracks; the remaining tracks, the data tracks, are used by CP/M 3 for file storage. Note that the system tracks are used by CP/M 3 only during system cold start and warm start. All other CP/M 3 disk access is directed to the data tracks of the disk. To maintain compatibility with Digital Research products, you should use an eight-inch, single-density, IBM® 3740 formatted disk with two system tracks.

## 1.7 Hardware Supported

You can customize the BIOS to match any hardware environment with the following general characteristics.

### 1.7.1 Hardware Supported by CP/M 3 Banked System

- Intel 8080, Intel 8085, or Zilog Z80 CPU or equivalent.
- A minimum of two and up to sixteen banks of memory with the top 4K-32K in common memory. Bank 1 must have contiguous memory from address 0000H to the base of common memory. A reasonable configuration consists of two banks of 48K RAM each, with the top 16K in common memory.
- One to sixteen disk drives of up to 512 megabytes capacity each.
- Some form of ASCII console device, usually a CRT.
- One to twelve additional character input and or output devices, such as printers, communications hardware, and plotters.

### 1.7.2 Hardware Supported by CP/M 3 Nonbanked System

- Intel 8080, Intel 8085, or Zilog Z80 CPU or equivalent.
- A minimum of 32K and up to 64K contiguous memory addressable from location zero.
- One to sixteen disk drives of up to 512 megabytes capacity each.
- Some form of ASCII console device, usually a CRT.
- One to twelve additional input and or output devices, usually including a printer.

Because most CP/M-compatible software is distributed on eight-inch, soft-sectored, single-density floppy disks, it is recommended that a CP/M 3 hardware configuration include a minimum of two disk drives, at least one of which is a single-density floppy disk drive.

## 1.8 Customizing CP/M 3

Digital Research supplies the BDOS files for a banked and a nonbanked version of CP/M 3. A system generation utility, GENCPM, is provided with CP/M 3 to create a version of the operating system tailored to your hardware. GENCPM combines the BDOS and your customized BIOS files to create a CPM3.SYS file, which is loaded into memory at system start-up. The CPM3.SYS file contains the BDOS and BIOS system components and information indicating where these modules reside in memory.

Digital Research supplies a CP/M 3 loader file, CPMLDR, which you can link with your customized loader BIOS and use to load the CPM3.SYS file into memory. CPMLDR is a small, self-contained version of CP/M 3 that supports only console output and sequential file input. Consistent with CP/M 3 organization, it contains two modules: an invariant CPMLDR_BDOS, and a variant CPMLDR_BIOS, which is adapted to match the host microcomputer hardware environment. The CPMLDR_BIOS module can perform cold start initialization of I/O ports and similar functions. CPMLDR can display a memory map of the CP/M 3 system at start-up. This is a GENCPM option.

The following steps tell you how to create a new version of CP/M 3 tailored to your specific hardware.

1) Write and assemble a customized BIOS following the specifications described in Section 3. This software module must correspond to the exact physical characteristics of the target system, including memory and port addresses, peripheral types, and drive characteristics.

2) Use the system generation utility, GENCPM, to create the CPM3.SYS file containing the CP/M 3 distributed BDOS and your customized BIOS, as described in Section 5.

3) Write a customized loader BIOS (LDRBIOS) to reside on the system tracks as part of CPMLDR. CPMLDR loads the CPM3.SYS file into memory from disk. Section 5 gives the instructions for customizing the LDRBIOS and generating CPMLDR. Link your customized LDRBIOS file with the supplied CPMLDR file.

4) Use the COPYSYS utility to put CPMLDR on the system tracks of a disk.

5) Test and debug your customized version of CP/M 3.

If you have banked memory, Digital Research recommends that you first use your customized BIOS to create a nonbanked version of the CP/M 3 operating system. You can leave your entire BIOS in common memory until you have a working system. Test all your routines in a nonbanked version of CP/M 3 before you create a banked version.

## 1.9 Initial Load (Cold Boot) of CP/M 3

CP/M 3 is loaded into memory as follows. Execution is initiated by a four-stage procedure. The first stage consists of loading into memory a small program, called the Cold Boot Loader, from the system tracks of the Boot disk. This load operation is typically handled by a hardware feature associated with system reset. The Cold Boot Loader is usually 128 or 256 bytes in length.

In the second stage, the Cold Boot Loader loads the memory image of the CP/M 3 system loader program, CPMLDR, from the system tracks of a disk into memory and passes control to it. For a banked system, the Cold Boot Loader loads CPMLDR into Bank 0. A PROM loader can perform stages one and two.

In the third stage, CPMLDR reads the CPM3.SYS file, which contains the BDOS and customized BIOS, from the the data area of the disk into the memory addresses assigned by GENCPM. In a banked system, CPMLDR reads the common part of the BDOS and BIOS into the common part of memory, and reads the banked part of the BDOS and BIOS into the area of memory below common_base in Bank 0. CPMLDR then transfers control to the Cold BOOT system initialization routine in the BIOS.

For the final stage, the BIOS Cold BOOT routine, BIOS Function 0, performs any remaining necessary hardware initialization, displays the sign-on message, and reads the CCP from the system tracks or from a CCP.COM file on disk into location 100H of the TPA. The Cold BOOT routine transfers control to the CCP, which then displays the system prompt.

Section 2 provides an overview of the organization of the System Control Block and the data structures and functions in the CP/M 3 BIOS.

End of Section 1

# Section 2
# CP/M 3 BIOS Overview

This section describes the organization of the CP/M 3 BIOS and the BIOS jump vector. It provides an overview of the System Control Block, followed by a discussion of system initialization procedures, character I/O, clock support, disk I/O, and memory selects and moves.

## 2.1 Organization of the BIOS

The BIOS is the CP/M 3 module that contains all hardware-dependent input and output routines. To configure CP/M 3 for a particular hardware environment, use the sample BIOS supplied with this document and adapt it to the specific hardware of the target system.

Alternatively, you can modify an existing CP/M 2.2 BIOS to install CP/M 3 on your target machine. Note that an unmodified CP/M 2.2 BIOS does not work with the CP/M 3 operating system. See Appendix C for a description of the modifications necessary to convert a CP/M 2.2 BIOS to a CP/M 3 BIOS.

The BIOS is a set of routines that performs system initialization, character-oriented I/O to the console and printer devices, and physical sector I/O to the disk devices. The BIOS also contains routines that manage block moves and memory selects for systems with bank-switched memory. The BIOS supplies tables that define the layout of the disk devices and allocate buffer space which the BDOS uses to perform record blocking and deblocking. The BIOS can maintain the system time and date in the System Control Block.

Table 2-1 describes the entry points into the BIOS from the Cold Start Loader and the BDOS. Entry to the BIOS is through a jump vector. The jump vector is a set of 33 jump instructions that pass program control to the individual BIOS subroutines.

You must include all of the entry points in the BIOS jump vector in your BIOS. However, if your system does not support some of the functions provided for in the BIOS, you can use empty subroutines for those functions. For example, if your system does not support a printer, JMP LIST can reference a subroutine consisting of only a RET instruction. Table 2-1 shows the elements of the jump vector.

**Table 2-1. CP/M 3 BIOS Jump Vector**

| No. | Instruction | Description |
|---|---|---|
| 0 | JMP BOOT | Perform cold start initialization |
| 1 | JMP WBOOT | Perform warm start initialization |
| 2 | JMP CONST | Check for console input character ready |
| 3 | JMP CONIN | Read Console Character in |
| 4 | JMP CONOUT | Write Console Character out |
| 5 | JMP LIST | Write List Character out |
| 6 | JMP AUXOUT | Write Auxiliary Output Character |
| 7 | JMP AUXIN | Read Auxiliary Input Character |
| 8 | JMP HOME | Move to Track 00 on Selected Disk |
| 9 | JMP SELDSK | Select Disk Drive |
| 10 | JMP SETTRK | Set Track Number |
| 11 | JMP SETSEC | Set Sector Number |
| 12 | JMP SETDMA | Set DMA Address |
| 13 | JMP READ | Read Specified Sector |
| 14 | JMP WRITE | Write Specified Sector |
| 15 | JMP LISTST | Return List Status |
| 16 | JMP SECTRN | Translate Logical to Physical Sector |
| 17 | JMP CONOST | Return Output Status of Console |
| 18 | JMP AUXIST | Return Input Status of Aux. Port |
| 19 | JMP AUXOST | Return Output Status of Aux. Port |
| 20 | JMP DEVTBL | Return Address of Char. I/O Table |
| 21 | JMP DEVINI | Initialize Char. I/O Devices |
| 22 | JMP DRVTBL | Return Address of Disk Drive Table |
| 23 | JMP MULTIO | Set Number of Logically Consecutive sectors to be read or written |
| 24 | JMP FLUSH | Force Physical Buffer Flushing for user-supported deblocking |
| 25 | JMP MOVE | Memory to Memory Move |
| 26 | JMP TIME | Time Set/Get signal |
| 27 | JMP SELMEM | Select Bank of Memory |
| 28 | JMP SETBNK | Specify Bank for DMA Operation |
| 29 | JMP XMOVE | Set Bank When a Buffer is in a Bank other than 0 or 1 |
| 30 | JMP USERF | Reserved for System Implementor |
| 31 | JMP RESERV1 | Reserved for Future Use |
| 32 | JMP RESERV2 | Reserved for Future Use |

Each jump address in Table 2-1 corresponds to a particular subroutine that performs a specific system operation. Note that two entry points are reserved for future versions of CP/M, and one entry point is provided for OEM subroutines, accessed only by direct BIOS calls using BDOS Function 50. Table 2-2 shows the five categories of system operations and the function calls that accomplish these operations.

**Table 2-2. CP/M 3 BIOS Functions**

| Operation | Function |
|---|---|
| System Initialization | |
| | BOOT, WBOOT, DEVTBL, DEVINI, DRVTBL |
| Character I/O | |
| | CONST, CONIN, CONOUT, LIST, AUXOUT, AUXIN, LISTST, CONOST, AUXIST, AUXOST |
| Disk I/O | |
| | HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, SECTRN, MULTIO, FLUSH |
| Memory Selects and Moves | |
| | MOVE, SELMEM, SETBNK, XMOVE |
| Clock Support | |
| | TIME |

You do not need to implement every function in the BIOS jump vector. However, to operate, the BDOS needs the BOOT, WBOOT, CONST, CONIN, CONOUT, HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, SECTRN, MULTIO, and FLUSH subroutines. Implement SELMEM and SETBNK only in a banked environment. You can implement MULTIO, FLUSH, and TIME as returns with a zero in register A. DEVICE and some other utilities use the remaining entry points, but it is not necessary to fully implement them in order to debug and develop the system.

**Note:** include all routines but make the nonimplemented routines a RET instruction.

## 2.2 System Control Block

The System Control Block (SCB) is a data structure located in the BDOS. The SCB is a communications area referenced by the BDOS, the CCP, the BIOS, and other system components. The SCB contains system parameters and variables, some of which the BIOS can reference. The fields of the SCB are named, and definitions of these names are supplied as public variable and subroutine names in the SCB.ASM file contained on the distribution disk. See Section 3.1 for a discussion of the System Control Block.

## 2.3 System Initialization

When the BOOT and WBOOT routines of the BIOS get control, they must initialize two system parameters in Page Zero of memory, as shown in Table 2-3.

**Table 2-3. Initialization of Page Zero**

| Location | Description |
|---|---|
| 0,1,2 | Set to JMP WBOOT (0000H: JMP BIOS+3). Location 1 and 2 must contain the address of WBOOT in the jump vector. |
| 5,6,7 | Set to JMP BDOS, the primary entry point to CP/M 3 for transient programs. The current address of the BDOS is maintained in the variable @MXTPA in the System Control Block. (See Section 3.1, "System Control Block," and BIOS Function 1: WBOOT on page 52.) |

The BOOT and WBOOT routine must load the CCP into the TPA in Bank 1 at location 0100H. The CCP can be loaded in two ways. If there is sufficient space on the system tracks, the CCP can be stored on the system tracks and loaded from there. If you prefer, or if there is not sufficient space on the system tracks, the BIOS Cold BOOT routine can read the CCP into memory from the file CCP.COM on disk.

If the CCP is in a .COM file, use the BOOT and WBOOT routines to perform any necessary system initialization, then use the BDOS functions to OPEN and READ the CCP.COM file into the TPA. In bank-switched systems, the CCP must be read into the TPA in Bank 1.

In bank-switched systems, your Cold BOOT routine can place a copy of the CCP into a reserved area of an alternate bank after loading the CCP into the TPA in Bank 1. Then the Warm BOOT routine can copy the CCP into the TPA in Bank 1 from the alternate bank, rather than reloading the CCP from disk, thus avoiding all disk accesses during warm starts.

There is a 128-byte buffer in the resident portion of the BDOS in a banked system that can be used by BOOT and WBOOT. The address of this buffer is stored in the SCB variable @BNKBF. BOOT and WBOOT can use this buffer when copying the CCP to and from the alternate bank.

The system tracks for CP/M 3 are usually partitioned as shown in the following figure:

| | | |
|---|---|---|
| Cold Start Ldr | CPMLDR | CCP (optional) |

**Figure 2-1. CP/M 3 System Tracks**

The cold start procedure is designed so you need to initialize the system tracks only once. This is possible because the system tracks contain the system loader and need not change when you change the CP/M 3 operating system. The Cold Start Loader loads CPMLDR into a constant memory location that is chosen when the system is configured. However, CPMLDR loads the BDOS and BIOS system components into memory as specified in the CPM3.SYS file generated by GENCPM, the system generation utility. Thus, CP/M 3 allows the user to configure a new system with GENCPM and then run it without having to update the system tracks of the system disk.

## 2.4 Character I/O

CP/M 3 assumes that all simple character I/O operations are performed in 8-bit ASCII, upper- and lowercase, with no parity. An ASCII CRTL-Z (1AH) denotes an end-of-file condition for an input device.

Table 2-4 lists the characteristics of the logical devices.

**Table 2-4. CP/M 3 Logical Device Characteristics**

| Device | Characteristics |
|---|---|
| CONIN, CONOUT | The interactive console that communicates with the operator, accessed by CONST, CONIN, CONOUT, and CONOUTST. Typically, the CONSOLE is a device such as a CRT or teletype, interfaced serially, but it can also be a memory-mapped video display and keyboard. The console is an input device and an output device. |
| LIST | The system printer, if it exists on your system. LIST is usually a hard-copy device such as a printer or teletypewriter. |
| AUXOUT | The auxiliary character output device, such as a modem. |
| AUXIN | The auxiliary character input device, such as a modem. |

Note that you can define a single peripheral as the LIST, AUXOUT, and AUXIN device simultaneously. If you assign no peripheral device as the LIST, AUXOUT, or AUXIN device, the AUXOUT and LIST routines can just return, and the AUXIN routine can return with a 1AH (CTRL-Z) in register A to indicate an immediate end-of-file.

CP/M 3 supports character device I/O redirection. This means that you can direct a logical device, such as CONIN or AUXOUT, to one or more physical devices. The DEVICE utility allows you to reassign devices and display, and to change the current device configurations, as described in the CP/M Plus User's Guide. The I/O redirection facility is optional. You should not implement it until the rest of your BIOS is fully functional.

## 2.5 Disk I/O

The BDOS accomplishes disk I/O by making a sequence of calls to the various disk access subroutines in the BIOS. The subroutines set up the disk number to access, the track and sector on a particular disk, and the Direct Memory Access (DMA) address and bank involved in the I/O operation. After these parameters are established, the BDOS calls the READ or WRITE function to perform the actual I/O operation.

Note that the BDOS can make a single call to SELDSK to select a disk drive, follow it with a number of read or write operations to the selected disk, and then select another drive for subsequent operations.

CP/M 3 supports multiple sector read or write operations to optimize rotational latency on block disk transfers. You can implement the multiple sector I/O facility in the BIOS by using the multisector count passed to the MULTIO entry point. The BDOS calls MULTIO to read or write up to 128 sectors. For every sector number 1 to n, the BDOS calls SETDMA then calls READ or WRITE.

Table 2-5 shows the sequence of BIOS calls that the BDOS makes to read or write a physical disk sector in a nonbanked and a banked system. Table 2-6 shows the sequence of calls the BDOS makes to the BIOS to read or write multiple contiguous physical sectors in a nonbanked and banked system.

**Table 2-5. BDOS Calls to BIOS in Nonbanked and Banked Systems**

| Call | Explanation |
|---|---|
| **Nonbanked BDOS** | |
| SELDSK | Called only when disk is initially selected or reselected. |
| SETTRK | Called for every read or write of a physical sector. |
| SETSEC | Called for every read or write of a physical sector. |
| SETDMA | Called for every read or write of a physical sector. |
| READ, WRITE | Called for every read or write of a physical sector. |
| **Banked BDOS** | |
| SELDSK | Called only when disk is initially selected or reselected. |
| SETTRK | Called for every read or write of a physical sector. |
| SETSEC | Called for every read or write of a physical sector. |
| SETDMA | Called for every read or write of a physical sector. |
| SETBNK | Called for every read or write of a physical sector. |
| READ, WRITE | Called for every read or write of a physical sector. |

**Table 2-6. Multiple Sector I/O in Nonbanked and Banked Systems**

| Call | Explanation |
|---|---|
| **Nonbanked BDOS** | |
| SELDSK | Called only when disk is initially selected or reselected. |
| MULTIO | Called to inform the BIOS that the next n calls to disk READ or disk WRITE require a transfer of n contiguous physical sectors to contiguous memory. |
| SETTRK | Called for every read or write of a physical sector. |
| SETSEC | Called for every read or write of a physical sector. |
| SETDMA | Called for every read or write of a physical sector. |
| READ, WRITE | Called for every read or write of a physical sector. |
| **Banked BDOS** | |
| SELDSK | Called only when disk is initially selected or reselected. |
| MULTIO | Called to inform the BIOS that the next n calls to disk READ or disk WRITE require a transfer of n contiguous physical sectors to contiguous memory. |
| SETTRK | Called for every read or write of a physical sector. |
| SETSEC | Called for every read or write of a physical sector. |
| SETDMA | Called for every read or write of a physical sector. |
| SETBNK | Called for every read or write of a physical sector. |
| READ, WRITE | Called for every read or write of a physical sector. |

Table 2-7 shows the sequence of BDOS calls to read two contiguous physical sectors in a banked system.

**Table 2-7. Reading Two Contiguous Sectors in Banked System**

| Call | Explanation |
|---|---|
| SELDSK | Called to initially select disk |
| MULTIO | With a value of 2 |
| SETTRK | For first sector |
| SETSEC | For first sector |
| SETDMA | For first sector |
| SETBNK | |
| READ | |
| SETTRK | For second sector |
| SETSEC | For second sector |
| SETDMA | For second sector |
| SETBNK | |
| READ | |

The CP/M 3 BDOS performs its own blocking and deblocking of logical 128-byte records. Unlike earlier versions of CP/M, the BIOS READ and WRITE routines always transfer physical sectors as specified in the Disk Parameter Block to or from the DMA buffer. The Disk Parameter Header defines one or more physical sector buffers which the BDOS uses for logical record blocking and deblocking.

In a banked environment, CP/M 3 maintains a cache of deblocking buffers and directory records using a Least Recently Used (LRU) buffering scheme. The LRU buffer is the first to be reused when the system runs out of buffer space. The BDOS maintains separate buffer pools for directory and data record caching.

The BIOS contains the data structures to control the data and directory buffers and the hash tables. You can either assign these buffers and tables yourself in the BIOS, or allow the GENCPM utility to generate them automatically.

Hash tables greatly speed directory searching. The BDOS can use hash tables to determine the location of directory entries and therefore reduce the number of disk accesses required to read a directory entry. The hash table allows the BDOS to directly access the sector of the directory containing the desired directory entry without having to read the directory sequentially. By eliminating a sequential read of the directory records, hashing also increases the percentage of time that the desired directory record is in a buffer, eliminating the need for any physical disk accesses in these cases. Hash tables and directory caches eliminate many of the directory accesses required when accessing large files. However, in a nonbanked system, hash tables increase the size of the operating system.

When the BIOS finds an error condition, the READ and WRITE routines should perform several retries before reporting the error condition to the BDOS. Ten retries are typical. If the BIOS returns an error condition to the BDOS, the BDOS reports the error to the user in the following form:

    CP/M Error on d: Disk I/O

The d: represents the drive specification of the relevant drive.

To provide better diagnostic capabilities for the user, it is often desirable to print a more explicit error message from the BIOS READ or WRITE routines before the BIOS returns an error code to the BDOS. The BIOS should interrogate the SCB Error Mode Variable to determine if it is appropriate to print a message on the console.

## 2.6 Memory Selects and Moves

Four BIOS functions are provided to perform memory management. The functions are MOVE, XMOVE, SELMEM, and SETBNK. The XMOVE, SELMEM, and SETBNK memory management routines are applicable to the BIOS of banked systems.

The BDOS uses the BIOS MOVE routine to perform memory-to-memory block transfers. In a banked system, the BDOS calls XMOVE to specify the source and destination banks to be used by the MOVE routine. If you use memory that is not in the common area for data record buffers, you must implement the XMOVE routine.

The BDOS uses SELMEM when the operating system needs to execute code or access data in other than the currently selected bank.

The BDOS calls the SETBNK routine prior to calling disk READ or disk WRITE functions. The SETBNK routine must save its specified bank as the DMA bank. When the BDOS invokes a disk I/O routine, the I/O routine should save the current bank number and select the DMA bank prior to the disk READ or WRITE. After completion of the disk READ or WRITE, the disk I/O routine must reselect the current bank. Note that when the BDOS calls the disk I/O routines, Bank 0 is in context (selected).

## 2.7 Clock Support

If the system has a real-time clock or is capable of keeping time, possibly by counting interrupts from a counter/timer chip, then the BIOS can maintain the time of day in the System Control Block and update the time on clock interrupts. BIOS Function 26 is provided for those systems where the clock is unable to generate an interrupt.

The time of day is kept as four fields. @DATE is a binary word containing the number of days since 31 December 1977. The bytes @HOUR, @MIN, and @SEC in the System Control Block contain the hour, minute, and second in Binary Coded Decimal (BCD) format.

End of Section 2

# Section 3
# CP/M 3 BIOS Functional Specifications

This section contains a detailed description of the CP/M 3 BIOS. The section first discusses the BIOS data structures and their relationships, including the System Control Block, the drive table, the Disk Parameter Header, the Disk Parameter Block, the Buffer Control Blocks, and the character I/O table. The overview of the data structures is followed by a summary of the functions in the BIOS jump vector. A detailed description of the entry values and returned values for each jump instruction in the BIOS jump vector follows the summary. The last part of this section discusses the steps to follow when assembling and linking your customized BIOS.

## 3.1 The System Control Block

The System Control Block (SCB) is a data structure located in the BDOS. The SCB contains flags and data used by the CCP, the BDOS, the BIOS, and other system components. The BIOS can access specific data in the System Control Block through the public variables defined in the SCB.ASM file, which is supplied on the distribution disk.

Declare the variable names you want to reference in the SCB as externals in your BIOS.ASM source file. Then link your BIOS with the SCB.REL module.

In the SCB.ASM file, the high-order byte of the various SCB addresses is defined as 0FEH. The linker marks absolute external equates as page relocatable when generating a System Page Relocatable (SPR) format file. GENCPM recognizes page relocatable addresses of 0FExxH as references to the System Control Block in the BDOS. GENCPM changes these addresses to point to the actual SCB in the BDOS when it is relocating the system.

Do not perform assembly-time arithmetic on any references to the external labels of the SCB. The result of the arithmetic could alter the page value to something other than 0FEH.

Listing 3-1 shows the SCB.ASM file. The listing shows the field names of the System Control Block. A @ before a name indicates that it is a data item. A ? preceding a name indicates that it is the label of an instruction. In the listing, r/w means Read-Write, and r/o means Read-Only. The BIOS can modify a Read-Write variable, but must not modify a Read-Only variable. Table 3-1 describes each item in the System Control Block in detail.

```
        title 'System Control Block Definition for CP/M3 BIOS'

        public @civec, @covec, @aivec, @aovec, @lovec, @bnkbf
        public @crdma, @crdsk, @vinfo, @resel, @fx, @usrcd
        public @mltio, @ermde, @erdsk, @media, @bflgs
        public @date, @hour, @min, @sec, ?erjmp, @mxtpa

scb$base equ    0FE00H          ; Base of the SCB

@CIVEC  equ     scb$base+22h    ; Console Input Redirection
                                ; Vector (word, r/w)
@COVEC  equ     scb$base+24h    ; Console Output Redirection
                                ; Vector (word, r/w)
@AIVEC  equ     scb$base+26h    ; Auxiliary Input Redirection
                                ; Vector (word, r/w)
@AOVEC  equ     scb$base+28h    ; Auxiliary Output Redirection
                                ; Vector (word, r/w)
@LOVEC  equ     scb$base+2Ah    ; List Output Redirection
                                ; Vector (word, r/w)
@BNKBF  equ     scb$base+35h    ; Address of 128 Byte Buffer
                                ; for Banked BIOS (word, r/o)
@CRDMA  equ     scb$base+3Ch    ; Current DMA Address
                                ; (word, r/o)
@CRDSK  equ     scb$base+3Eh    ; Current Disk (byte, r/o)
@VINFO  equ     scb$base+3Fh    ; BDOS Variable "INFO"
                                ; (word, r/o)
@RESEL  equ     scb$base+41h    ; FCB Flag (byte, r/o)
@FX     equ     scb$base+43h    ; BDOS Function for Error
                                ; Messages (byte, r/o)
@USRCD  equ     scb$base+44h    ; Current User Code (byte, r/o)
@MLTIO  equ     scb$base+4Ah    ; Current Multisector Count
                                ; (byte,r/w)
@ERMDE  equ     scb$base+4Bh    ; BDOS Error Mode (byte, r/o)
@ERDSK  equ     scb$base+51h    ; BDOS Error Disk (byte, r/o)
@MEDIA  equ     scb$base+54h    ; Set by BIOS to indicate
                                ; open door (byte,r/w)
@BFLGS  equ     scb$base+57h    ; BDOS Message Size Flag
                                ; (byte,r/o)
@DATE   equ     scb$base+58h    ; Date in Days Since 1 Jan 78
                                ; (word, r/w)
@HOUR   equ     scb$base+5Ah    ; Hour in BCD (byte, r/w)
@MIN    equ     scb$base+5Bh    ; Minute in BCD (byte, r/w)
@SEC    equ     scb$base+5Ch    ; Second in BCD (byte, r/w)
?ERJMP  equ     scb$base+5Fh    ; BDOS Error Message Jump
                                ; (3 bytes, r/w)
@MXTPA  equ     scb$base+62h    ; Top of User TPA
                                ; (address at 6,7)(word, r/o)
        end
```

**Listing 3-1. SCB.ASM File**

The following table describes in detail each of the fields of the System Control Block.

**Table 3-1. System Control Block Fields**

| Field | Meaning |
|---|---|
| @CIVEC, @COVEC, @AIVEC, @AOVEC, @LOVEC (Read-Write Variable) | These fields are the 16 bit I/O redirection vectors for the five logical devices: console input, console output, auxiliary input, auxiliary output, and the list device. (See Section 3.4.2, "Character I/O Functions.") |
| @BNKBF (Read-Only Variable) | @BNKBF contains the address of a 128 byte buffer in the resident portion of the BDOS in a banked system. This buffer is available for use during BOOT and WBOOT only. You can use it to transfer a copy of the CCP from an image in an alternate bank if the system does not support interbank moves. |
| @CRDMA, @FX, @USRCD, @ERDSK (Read-Only Variable) | These variables contain the current DMA address, the BDOS function number, the current user code, and the disk code of the drive on which the last error occurred. They can be displayed when a BDOS error is intercepted by the BIOS. See ?ERJMP. |
| @CRDSK (Read-Only Variable) | @CRDSK is the current default drive, set by BDOS Function 14. |
| @VINFO, @RESEL (Read-Only Variable) | If @RESEL is equal to 0FFH then @VINFO contains the address of a valid FCB. If @RESEL is not equal to 0FFH, then @VINFO is undefined. You can use @VINFO to display the filespec when the BIOS intercepts a BDOS error. |

**Table 3-1. (continued)**

| Field | Meaning |
|---|---|
| @MLTIO | (Read-Write Variable)<br><br>@MLTIO contains the current multisector count. The BIOS can change the multisector count directly, or through BDOS Function 44. The value of the multisector count can range from 1 to 128. |
| @ERMDE | (Read-Only Variable)<br><br>@ERMDE contains the current BDOS error mode. 0FFH indicates the BDOS is returning error codes to the application program without displaying any error messages. 0FEH indicates the BDOS is both displaying and returning errors. Any other value indicates the BDOS is displaying errors without notifying the application program. |
| @MEDIA | (Read-Write Variable)<br><br>@MEDIA is global system flag indicating that a drive door has been opened. The BIOS routine that detects the open drive door sets this flag to 0FFH. The BIOS routine also sets the MEDIA byte in the Disk Parameter Header associated with the open-door drive to 0FFH. |
| @BFLGS | (Read-Only Variable)<br><br>The BDOS in CP/M 3 produces two kinds of error messages: short error messages and extended error messages. Short error messages display one or two lines of text. Long error messages display a third line of text containing the filename, filetype, and BDOS Function Number involved in the error.<br><br>In banked systems, GENCPM sets this flag in the System Control Block to indicate whether the BIOS displays short or extended error messages. Your error message handler should check this byte in the System Control Block. If the high-order bit, bit 7, is set to 0, the BDOS displays short error messages. If the high-order bit is set to 1, the BDOS displays the extended three-line error messages. |

**Table 3-1. (continued)**

| Field | Meaning |
|---|---|
| @BFLGS (continued) | For example, the BDOS displays the following error message if the BIOS returns an error from READ and the BDOS is displaying long error messages.<br><br>CP/M Error on d: Disk I/O<br>BDOS Function = nn File = filename.typ<br><br>In the above error message, Function nn and filename.typ represent BDOS function number and file specification involved, respectively. |
| @DATE (Read-Write Variable) | The number of days since 31 December 1977, expressed as a 16-bit unsigned integer, low byte first. A real-time clock interrupt can update the @DATE field to indicate the current date. |
| @HOUR, @MIN, @SEC (Read-Write Variable) | These 2-digit Binary Coded Decimal (BCD) fields indicate the current hour, minute, and second if updated by a real-time clock interrupt. |
| ?ERJMP (Read-Write Code Label) | The BDOS calls the error message subroutine through this jump instruction. Register C contains an error code as follows:<br><br>1 Permanent Error<br>2 Read Only Disk<br>3 Read Only File<br>4 Select Error<br>7 Password Error<br>8 File Exists<br>9 ? in Filename<br><br>Error code 1 above results in the BDOS message Disk I/O. |

**Table 3-1. (continued)**

| Field | Meaning |
|---|---|
| ?ERJMP (continued) | |
| | The ?ERJMP vector allows the BIOS to intercept the BDOS error messages so you can display them in a foreign language. Note that this vector is not branched to if the application program is expecting return codes on physical errors. Refer to the CP/M Plus Programmer's Guide for more information.<br><br>?ERJMP is set to point to the default (English) error message routine contained in the BDOS. The BOOT routine can modify the address at ?ERJMP+1 to point to an alternate message routine. Your error message handler can refer to @FX, @VINFO (if @RESEL is equal to 0FFH), @CRDMA, @CRDSK, and @USRCD to print additional error information. Your error handler should return to the BDOS with a RET instruction after printing the appropriate message. |
| @MXTPA (Read-Only Variable) | |
| | @MXTPA contains the address of the current BDOS entry point. This is also the address of the top of the TPA. The BOOT and WBOOT routines of the BIOS must use this address to initialize the BDOS entry JMP instruction at location 005H, during system initialization. Each time a RSX is loaded, @MXTPA is adjusted by the system to reflect the change in the available User Memory (TPA). |

## 3.2 Character I/O Data Structures

The BIOS data structure CHRTBL is a character table describing the physical I/O devices. CHRTBL contains 6-byte physical device names and the characteristics of each physical device. These characteristics include a mode byte, and the current baud rate, if any, of the device. The DEVICE utility references the physical devices through the names and attributes contained in your CHRTBL. DEVICE can also display the physical names and characteristics in your CHRTBL.

The mode byte specifies whether the device is an input or output device, whether it has a selectable baud rate, whether it is a serial device, and if XON/XOFF protocol is enabled.

Listing 3-2 shows a sample character device table that the DEVICE utility uses to set and display I/O direction.

```
        ; sample character device table

chrtbl  db 'CRT   '                 ; console VDT
        db mb$in$out+mb$serial+mb$soft$baud
        db baud$9600

        db 'LPT   '                 ; system serial printer
        db mb$output+mb$serial+mb$soft$baud+mb$xon
        db baud$9600

        db 'TI810 '                 ; alternate printer
        db mb$output+mb$serial+mb$soft$baud
        db baud$9600

        db 'MODEM '                 ; 300 baud modem port
        db mb$in$out+mb$serial+mb$soft$baud
        db baud$300

        db 'VAX   '                 ; interface to VAX 11/780
        db mb$in$out+mb$serial+mb$soft$baud
        db baud$9600

        db 'DIABLO'                 ; Diablo 630 daisy wheel printer
        db mb$output+mb$serial+mb$soft$baud+mb$xon$xoff
        db baud$1200

        db 'CEN   '                 ; centronics type parallel printer
        db mb$output
        db baud$none

        db 0                        ; table terminator
```

**Listing 3-2. Sample Character Device Table**

Listing 3-3 shows the equates for the fields contained in the sample character device table. Many systems do not support all of these baud rates.

```
        ; equates for mode byte fields

mb$input                equ 0000$0001b  ; device may do input
mb$output               equ 0000$0010b  ; device may do output
mb$in$out               equ mb$input+mb$output ; dev may do both
mb$soft$baud            equ 0000$0100b  ; software selectable
                                        ; baud rates
mb$serial               equ 0000$1000b  ; device may use protocol
mb$xon$xoff             equ 0001$0000b  ; XON/XOFF protocol
                                        ; enabled


        ; equates for baud rate byte

baud$none               equ 0           ; no baud rate
                                        ; associated with device
baud$50                 equ 1           ; 50 baud
baud$75                 equ 2           ; 75 baud
baud$110                equ 3           ; 110 baud
baud$134                equ 4           ; 134.5 baud
baud$150                equ 5           ; 150 baud
baud$300                equ 6           ; 300 baud
baud$600                equ 7           ; 600 baud
baud$1200               equ 8           ; 1200 baud
baud$1800               equ 9           ; 1800 baud
baud$2400               equ 10          ; 2400 baud
baud$3600               equ 11          ; 3600 baud
baud$4800               equ 12          ; 4800 baud
baud$7200               equ 13          ; 7200 baud
baud$9600               equ 14          ; 9600 baud
baud$19200              equ 15          ; 19.2k baud
```

**Listing 3-3. Equates for Mode Byte Bit Fields**

### 3.3 BIOS Disk Data Structures

The BIOS includes tables that describe the particular characteristics of the disk subsystem used with CP/M 3. This section describes the elements of these tables.

In general, each disk drive has an associated Disk Parameter Header (DPH) that contains information about the disk drive and provides a scratchpad area for certain BDOS operations. One of the elements of this Disk Parameter Header is a pointer to the Disk Parameter Block (DPB), which contains the actual disk description.

In the banked system, only the Disk Parameter Block must reside in common memory. The DPHs, checksum vectors, allocation vectors, Buffer Control Blocks, and Directory Buffers can reside in common memory or Bank 0. The hash tables can reside in common memory or any bank except Bank 1. The data buffers can reside in banked memory if you implement the XMOVE function.

Figure 3-1 shows the relationships between the drive table, the Disk Parameter Header, and the Data and Directory Buffer Control Block fields and their respective data structures and buffers.

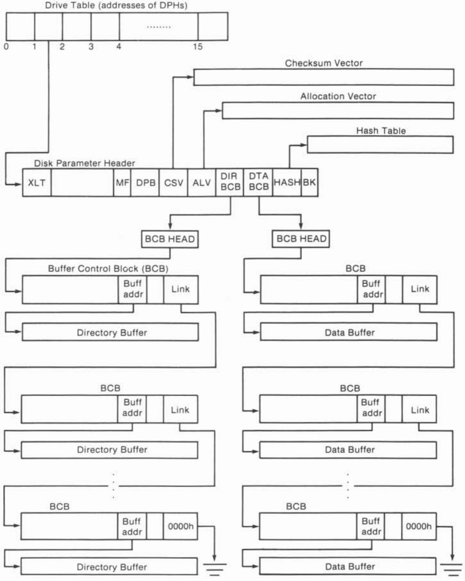


**Figure 3-1. Disk Data Structures in a Banked System**

### 3.3.1 Drive Table

The drive table consists of 16 words containing the addresses of the Disk Parameter Headers for each logical drive name, A through P, and takes the general form:

```
drivetable dw    dph0
           dw    dph1
           dw    dph2
           .
           .
           .
           dw    dphF
```

If a logical drive does not exist in your system, the corresponding entry in the drive table must be zero.

The GENCPM utility accesses the drive table to locate the various disk parameter data structures, so that it can determine which system configuration to use, and optionally allocate the various buffers itself. You must supply a drive table if you want GENCPM to do this allocation. If certain addresses in the Disk Parameter Headers referenced by this drive table are set to 0FFFEH, GENCPM allocates the appropriate data structures and updates the DPH. You can supply the drive table even if you have performed your own memory allocation. See the BIOS DRVTBL function described in Section 3.4.1.

### 3.3.2 Disk Parameter Header

In Figure 3-2, which shows the format of the Disk Parameter Header, b refers to bits.

| XLT | -0- | MF | DPB | CSV | ALV | DIRBCB | DTABCB | HASH | HBANK. |
|---|---|---|---|---|---|---|---|---|---|
| 16b | 72b | 8b | 16b | 16b | 16b | 16b | 16b | 16b | 8b |

**Figure 3-2. Disk Parameter Header Format**

Table 3-2 describes the fields of the Disk Parameter Header.

**Table 3-2. Disk Parameter Header Fields**

| Field | Comments |
|---|---|
| XLT | Set the XLT field to the address of the logical to physical sector translation table. If there is no sector translation and the logical and physical sector numbers are the same, set XLT to 0000H. Disk drives with identical sector skew factors can share the same translation table.<br><br>XLT is the value passed to SECTRN in registers DE. Usually the translation table consists of one byte per physical sector. Generally, it is advisable to keep the number of physical sectors per logical track to a reasonable value to prevent the translation table from becoming too large. In the case of disks with multiple heads, you can compute the head number from the track address rather than the sector address. |
| -0- | These 72 bits (9 bytes) of zeroes are the scratch area the BDOS uses to maintain various parameters associated with the drive. |
| MF | MF is the Media Flag. The BDOS resets MF to zero when the drive is logged in. The BIOS can set this flag and @MEDIA in the SCB to 0FFH if it detects that a drive door has been opened. If the flag is set to 0FFH, the BDOS checks for a media change prior to performing the next BDOS file operation on that drive. If the BDOS determines that the drive contains a new volume, the BDOS performs a login on that drive, and resets the MF flag to 00H. Note that the BDOS checks this flag only when a system call is made, and not during an operation. Usually, this flag is used only by systems that support door-open interrupts. |
| DPB | Set the DPB field to the address of a Disk Parameter Block that describes the characteristics of the disk drive. Several Disk Parameter Headers can address the same Disk Parameter Block if their drive characteristics are identical. (The Disk Parameter Block is described in Section 3.3.3.) |

**Table 3-2. (continued)**

| Field | Comments |
|---|---|
| CSV | CSV is the address of a scratchpad area used to detect changed disks. This address must be different for each removable media Disk Parameter Header. There must be one byte for every 4 directory entries (or 128 bytes of directory). In other words, length(CSV) = (DRM/4)+1. (See Table 3-3 for an explanation of the DRM field.) If the drive is permanently mounted, set the CKS variable in the DPB to 8000H and set CSV to 0000H. This way, no storage is reserved for a checksum vector. The checksum vector may be located in common memory or in Bank 0. Set CSV to 0FFFEH for GENCPM to set up the checksum vector. |
| ALV | ALV is the address of the scratchpad area called the allocation vector, which the BDOS uses to keep disk storage allocation information. This area must be unique for each drive.<br><br>The allocation vector usually requires 2 bits for each block on the drive. Thus, length(ALV) = (DSM/4) + 2. (See Table 3-3 for an explanation of the DSM field.) In the nonbanked version of CP/M 3, you can optionally specify that GENCPM reserve only one bit in the allocation vector per block on the drive. In this case, length(ALV) = (DSM/8) + 1.<br><br>The GENCPM option to use single-bit allocation vectors is provided in the nonbanked version of CP/M 3 because additional memory is required by the double-bit allocation vector. This option applies to all drives on the system.<br><br>With double-bit allocation vectors, CP/M 3 automatically frees, at every system warm start, all file blocks that are not permanently recorded in the directory. Note that file space allocated to a file is not permanently recorded in a directory unless the file is closed. Therefore, the allocation vectors in memory can indicate that space is allocated although directory records indicate that space is free for allocation. With single-bit allocation vectors, CP/M 3 requires that a drive be reset before this space can be reclaimed. Because it increases performance, CP/M 3 does not reset disks at system warm start. Thus, with single-bit allocation vectors, if you do not reset the disk system, DIR and SHOW can report an inaccurate amount of free space. With single-bit |

**Table 3-2. (continued)**

| Field | Comments |
|---|---|
| ALV (continued) | allocation vectors, the user must type a CTRL-C at the system prompt to reset the disk system to ensure accurate reporting of free space. Set ALV to 0FFFEH for GENCPM to automatically assign space for the allocation vector, single- or double-bit, during system generation. In the nonbanked system, GENCPM prompts for the type of allocation vector. In the banked system, the allocation vector is always double-bit and can reside in common memory or Bank 0. When GENCPM automatically assigns space for the allocation vector (ALV = 0FFFEH), it places the allocation vector in Bank 0. |
| DIRBCB | Set DIRBCB to the address of a single directory Buffer Control Block (BCB) in an unbanked system. Set DIRBCB to the address of a BCB list head in a banked system.<br><br>Set DIRBCB to 0FFFEH for GENCPM to set up the DIRBCB field. The BDOS uses directory buffers for all accesses of the disk directory. Several DPHs can refer to the same directory BCB or BCB list head; or, each DPH can reference an independent BCB or BCB list head. Section 3.3.4 describes the format of the Buffer Control Block. |
| DTABCB | Set DTABCB to the address of a single data BCB in an unbanked system. Set DTABCB to the address of a data BCB list head in a banked system.<br><br>Set DTABCB to 0FFFEH for GENCPM to set up the DTABCB field. The BDOS uses data buffers to hold physical sectors so that it can block and deblock logical 128-byte records. If the physical record size of the media associated with a DPH is 128 bytes, you can set the DTABCB field of the DPH to 0FFFFH, because in this case, the BDOS does not use a data buffer. |
| HASH | HASH contains the address of the optional directory hashing table associated with a DPH. Set HASH to 0FFFFH to disable directory hashing. |

**Table 3-2. (continued)**

| Field | Comments |
|---|---|
| HASH (continued) | Set HASH to 0FFFEH to make directory hashing on the drive a GENCPM option. Each DPH using hashing must reference a unique hash table. If a hash table is supplied, it must be 4*(DRM+1) bytes long, where DRM is one less than the length of the directory. In other words, the hash table must contain four bytes for each directory entry of the disk. |
| HBANK | Set HBANK to the bank number of the hash table. HBANK is not used in unbanked systems and should be set to zero. The hash tables can be contained in the system bank, common memory, or any alternate bank except Bank 1, because hash tables cannot be located in the Transient Program Area. GENCPM automatically sets HBANK when HASH is set to 0FFFEH. |

### 3.3.3 Disk Parameter Block

Figure 3-3 shows the format of the Disk Parameter Block, where b refers to bits.

| SPT | BSH | BLM | EXM | DSM | DRM | AL0 | AL1 | CKS | OFF | PSH | PHM |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 16b | 8b | 8b | 8b | 16b | 16b | 8b | 8b | 16b | 16b | 8b | 8b |

**Figure 3-3. Disk Parameter Block Format**

Table 3-3 describes the fields of the Disk Parameter Block.

**Table 3-3. Disk Parameter Block Fields**

| Field | Comments |
|---|---|
| SPT | Set SPT to the total number of 128-byte logical records per track. |
| BSH | Data allocation block shift factor. The value of BSH is determined by the data block allocation size. |
| BLM | Block mask. The value of BLM is determined by the data block allocation size. |

**Table 3-3. (continued)**

| Field | Comments |
|---|---|
| EXM | Extent mask determined by the data block allocation size and the number of disk blocks. |
| DSM | Determines the total storage capacity of the disk drive. DSM is one less than the total number of blocks on the drive. |
| DRM | Total number of directory entries minus one that can be stored on this drive. The directory requires 32 bytes per entry. |
| AL0, AL1 | Determine reserved directory blocks. See Figure 3-4 for more information. |
| CKS | The size of the directory check vector, (DRM/4)+1. Set bit 15 of CKS to 1 if the drive is permanently mounted. Set CKS to 8000H to indicate that the drive is permanently mounted and directory checksumming is not required.<br><br>**Note:** full directory checksumming is required on removable media to support the automatic login feature of CP/M 3. |
| OFF | The number of reserved tracks at the beginning of the logical disk. OFF is the track on which the directory starts. |
| PSH | Specifies the physical record shift factor. |
| PHM | Specifies the physical record mask. |

CP/M allocates disk space in a unit called a block. Blocks are also called allocation units, or clusters. BLS is the number of bytes in a block. The block size can be 1024, 2048, 4096, 8192, or 16384 (decimal) bytes.

A large block size decreases the size of the allocation vectors but can result in wasted disk space. A smaller block size increases the size of the allocation vectors because there are more blocks on the same size disk.

There is a restriction on the block size. If the block size is 1024, there cannot be more than 255 blocks present on a logical drive. In other words, if the disk is larger than 256K, it is necessary to use at least 2048 byte blocks.

The value of BLS is not a field in the Disk Parameter Block; rather, it is derived from the values of BSH and BLM as given in Table 3-4.

**Table 3-4. BSH and BLM Values**

| BLS | BSH | BLM |
|---|---|---|
| 1,024 | 3 | 7 |
| 2,048 | 4 | 15 |
| 4,096 | 5 | 31 |
| 8,192 | 6 | 63 |
| 16,384 | 7 | 127 |

The block mask, BLM, equals one less than the number of 128-byte records in an allocation unit, (BLS/128 - 1), or (2**BSH)-1.

The value of the Block Shift Factor, BSH, is determined by the data block allocation size. The Block Shift Factor (BSH) equals the logarithm base two of the block size in 128-byte records, or LOG2(BLS/128), where LOG2 represents the binary logarithm function.

The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255, as shown in Table 3-5.

**Table 3-5. Maximum EXM Values**

| BLS | EXM values | |
|---|---|---|
| | DSM<256 | DSM>255 |
| 1,024 | 0 | N/A |
| 2,048 | 1 | 0 |
| 4,096 | 3 | 1 |
| 8,192 | 7 | 3 |
| 16,384 | 15 | 7 |

The value of EXM is one less than the maximum number of 16K extents per FCB.

Set EXM to zero if you want media compatibility with an extended CP/M 1.4 system. This only applies to double-density CP/M 1.4 systems, with disk sizes greater than 256K bytes. It is preferable to copy double-density 1.4 disks to single-density, then reformat them and recreate them with the CP/M 3 system, because CP/M 3 uses directory entries more effectively than CP/M 1.4.

DSM is one less than the total number of blocks on the drive. DSM must be less than or equal to 7FFFH. If the disk uses 1024 byte blocks (BSH=3, BLM=7), DSM must be less than or equal to 00FFH. The product BLS*(DSM+1) is the total number of bytes the drive holds and must be within the capacity of the physical disk. It does not include the reserved operating system tracks.

The DRM entry is one less than the total number of 32-byte directory entries, and is a 16-bit value. DRM must be less than or equal to (BLS/32 * 16) - 1. DRM determines the values of AL0 and AL1. The two fields AL0 and AL1 can together be considered a string of 16 bits, as shown in Figure 3-4.

| AL0 | | | | | | | | AL1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 |

**Figure 3-4. AL0 and AL1**

Position 00 corresponds to the high-order bit of the byte labeled AL0, and position 15 corresponds to the low-order bit of the byte labeled AL1. Each bit position reserves a data block for a number of directory entries, thus allowing a maximum of 16 data blocks to be assigned for directory entries. Bits are assigned starting at 00 and filled to the right until position 15. AL0 and AL1 overlay the first two bytes of the allocation vector for the associated drive. Table 3-6 shows DRM maximums for the various block sizes.

**Table 3-6. BLS and Number of Directory Entries**

| BLS | Directory Entries | Maximum DRM |
|---|---|---|
| 1,024 | 32 * reserved blocks | 511 |
| 2,048 | 64 * reserved blocks | 1,023 |
| 4,096 | 128 * reserved blocks | 2,047 |
| 8,192 | 256 * reserved blocks | 4,095 |
| 16,384 | 512 * reserved blocks | 8,191 |

If DRM = 127 (128 directory entries), and BLS = 1024, there are 32 directory entries per block, requiring 4 reserved blocks. In this case, the 4 high-order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H. The maximum directory allocation is 16 blocks where the block size is determined by BSH and BLM.

The OFF field determines the number of tracks that are skipped at the beginning of the physical disk. It can be used as a mechanism for skipping reserved operating system tracks, which on system disks contain the Cold Boot Loader, CPMLDR, and possibly the CCP. It is also used to partition a large disk into smaller segmented sections.

PSH and PHM determine the physical sector size of the disk. All disk I/O is in terms of the physical sector size. Set PSH and PSM to zero if the BIOS is blocking and deblocking instead of the BDOS.

PSH specifies the physical record shift factor, ranging from 0 to 5, corresponding to physical record sizes of 128, 256, 512, 1K, 2K, or 4K bytes. It is equal to the logarithm base two of the physical record size divided by 128, or LOG2(sector_size/128). See Table 3-7 for PSH values.

PHM specifies the physical record mask, ranging from 0 to 31, corresponding to physical record sizes of 128, 256, 512, 1K, 2K, or 4K bytes. It is equal to one less than the sector size divided by 128, or, (sector_size/128)-1. See Table 3-7 for PHM values.

**Table 3-7. PSH and PHM Values**

| Sector size | PSH | PHM |
|---|---|---|
| 128 | 0 | 0 |
| 256 | 1 | 1 |
| 512 | 2 | 3 |
| 1,024 | 3 | 7 |
| 2,048 | 4 | 15 |
| 4,096 | 5 | 31 |

### 3.3.4 Buffer Control Block

A Buffer Control Block (BCB) locates physical record buffers for the BDOS. The BDOS uses the BCB to manage the physical record buffers during processing. More than one Disk Parameter Header can specify the same BCB. The GENCPM utility can create the Buffer Control Block.

Note that the BANK and LINK fields of the Buffer Control Block are present only in the banked system. Therefore, the Buffer Control Block is twelve bytes long in the nonbanked system, and fifteen bytes long in the banked system. Note also that only the DRV, BUFFAD, BANK, and LINK fields need to contain initial values. In Figure 3-5, which shows the form of the Buffer Control Block, b refers to bits.

| DRV | REC# | WFLG | 00 | TRACK | SECTOR | BUFFAD | BANK | LINK |
|---|---|---|---|---|---|---|---|---|
| 8b | 24b | 8b | 8b | 16b | 16b | 16b | 8b | 16b |

**Figure 3-5. Buffer Control Block Format**

h

Table 3-8 describes the fields of each Buffer Control Block.

**Table 3-8. Buffer Control Block Fields**

| Field | Comment |
|---|---|
| DRV | Identifies the disk drive associated with the record contained in the buffer located at address BUFFAD. If you do not use GENCPM to allocate buffers, you must set the DRV field to OFFH. |
| REC# | Identifies the record position of the current contents of the buffer located at address BUFFAD. REC# consists of the absolute sector number of the record where the first record of the directory is zero. |
| WFLG | Set by the BDOS to 0FFH to indicate that the buffer contains new data that has not yet been written to disk. When the data is written, the BDOS sets the WFLG to zero to indicate the buffer is no longer dirty. |
| 00 | Scratch byte used by BDOS. |
| TRACK | Contains the physical track location of the contents of the buffer. |
| SECTOR | Contains the physical sector location of the contents of the buffer. |
| BUFFAD | Specifies the address of the buffer associated with this BCB. |
| BANK | Contains the bank number of the buffer associated with this BCB. This field is only present in banked systems. |
| LINK | Contains the address of the next BCB in a linked list, or zero if this is the last BCB in the linked list. The LINK field is present only in banked systems. |

The BDOS distinguishes between two kinds of buffers: data buffers referenced by DTABCB, and directory buffers referenced by DIRBCB. In a banked system, the DIRBCB and DTABCB fields of a Disk Parameter Header each contain the address of a BCB list head rather than the address of an actual BCB. A BCB list head is a word containing the address of the first BCB in a linked list. If several DPHs reference the same BCB list, they must reference the same BCB list head. Each BCB has a LINK field that contains the address of the next BCB in the list, or zero if it is the last BCB.

In banked systems, the one-byte BANK field indicates the bank in which the data buffers are located. The BANK field of directory BCBs must be zero because directory buffers must be located in Bank 0, usually below the banked BDOS module, or in common memory. The BANK field is for systems that support direct memory-to-memory transfers from one bank to another. (See the BIOS XMOVE entry point in Section 3.4.4.)

The BCB data structures in a banked system must reside in Bank 0 or in common memory. The buffers of data BCBs can be located in any bank except Bank 1 (the Transient Program Area).

For banked systems that do not support interbank block moves through XMOVE, the BANK field must be set to 0 and the data buffers must reside in common memory. The directory buffers can be in Bank 0 even if the system does not support bank-to-bank moves.

In the nonbanked system, the DPH, DIRBCB, and DTABCB can point to the same BCB if the DPH defines a fixed media device. For devices with removable media, the DPH DIRBCB and the DPH DTABCB must reference different BCBs. In banked systems, the DPH DIRBCB and DTABCB must point to separate list heads.

In general, you can enhance the performance of CP/M 3 by allocating more BCBs, but the enhancement reduces the amount of TPA memory in nonbanked systems.

If you set the DPH DIRBCB or the DPH DTABCB fields to 0FFFEH, the GENCPM utility creates BCBs, allocates physical record buffers, and sets these fields to the address of the BCBs. This allows you to write device drivers without regard to buffer requirements.

### 3.3.5 Data Structure Macro Definitions

Several macro definitions are supplied with CP/M 3 to simplify the creation of some of the data structures in the BIOS. These macros are defined in the library file CPM3.LIB on the distribution disk.

To reference these macros in your BIOS, include the following statement:

```
        MACLIB    CPM3
```

DTBL Macro

Use the DTBL macro to generate the drive table, DRVTBL. It has one parameter, a list of the DPHs in your system. The list is enclosed in angle brackets.

The form of the DTBL macro call is

```
label: DTBL      <DPHA,DPHB,...,DPHP>
```

where DPHA is the address of the DPH for drive A, DPHB is the address of the DPH for drive B, up to drive P. For example,

```
DRVTBL:  DTBL    <ACSHD0,FDSD0,FDSD1>
```

This example generates the drive table for a three-drive system. The DTBL macro always generates a sixteen-word table, even if you supply fewer DPH names. The unused entries are set to zero to indicate the corresponding drives do not exist.

DPH Macro

The DPH macro routine generates a Disk Parameter Header (DPH). It requires two parameters: the address of the skew table for this drive, and the address of the Disk Parameter Block (DPB). Two parameters are optional: the maximum size of the checksum vector, and the maximum size of the allocation vector. If you omit the maximum size of the checksum vector and the maximum size of the allocation vector from the DPH macro invocation, the corresponding fields of the Disk Parameter Header are set to 0FFFEH so that GENCPM automatically allocates the vectors.

The form of the DPH macro call is

```
label:  DPH      ?trans,?dpb,[?csize],[?asize]
```

where:

| | |
|---|---|
| ?trans | is the address of the translation vector for this drive; |
| ?dpb | is the address of the DPB for this drive; |
| ?csize | is the maximum size in bytes of the checksum vector; |
| ?asize | is the maximum size in bytes of the allocation vector. |

The following example, which includes all four parameters, shows a typical DPH macro invocation for a standard single-density disk drive:

```
FDSD0:  DPH     SKEW6,DPB$SD,16,31
```

## SKEW Macro

The SKEW macro generates a skew table and requires the following parameters: the number of physical sectors per track, the skew factor, and the first sector number on each track (usually 0 or 1).

The form of the SKEW macro call is

```
label:  SKEW    ?secs,?skf,?fsc
```

where:

```
?secs  is the number of physical sectors per track;
?skf   is the sector skew factor;
?fsc   is the first sector number on each track.
```

The following macro invocation generates the skew table for a standard single-density disk drive.

```
SKEW6:  SKEW    26,6,1
```

## DPB Macro

The DPB macro generates a Disk Parameter Block specifying the characteristics of a drive type. It requires six parameters: the physical sector size in bytes, the number of physical sectors per track, the total number of tracks on the drive, the size of an allocation unit in bytes, the number of directory entries desired, and the number of system tracks to reserve at the beginning of the drive. There is an optional seventh parameter that defines the CKS field in the DPB. If this parameter is missing, CKS is calculated from the directory entries parameter.

The form of the DPB macro call is

```
label: DPB    ?psize,?pspt,?trks,?bls,?ndirs,?off[,?ncks]
```

where:

```
?psize  is the physical sector size in bytes;
?pspt   is the number of physical sectors per track;
?trks   is the number of tracks on the drive;
?bls    is the allocation unit size in bytes;
?ndirs  is the number of directory entries;
?off    is the number of tracks to reserve;
?ncks   is the number of checked directory entries.
```

The following example shows the parameters for a standard single-density disk drive:

```
DPB$SD: DPB     128,26,77,1024,64,2
```

The DPB macro can be used only when the disk drive is under eight megabytes. DPBs for larger disk drives must be constructed by hand.

### 3.4 BIOS Subroutine Entry Points

This section describes the entry parameters, returned values, and exact responsibilities of each BIOS entry point in the BIOS jump vector. The routines are arranged by function. Section 3.4.1 describes system initialization. Section 3.4.2 presents the character I/O functions, followed by Section 3.4.3, discussing the disk I/O functions. Section 3.4.4 discusses the BIOS memory select and move functions. The last section, 3.4.5, discusses the BIOS clock support function. Table 3-9 shows the BIOS entry points the BDOS calls to perform each of the four categories of system functions.

**Table 3-9. Functional Organization of BIOS Entry Points**

| Operation | Function |
|---|---|
| System Initialization | |
| | BOOT, WBOOT, DEVTBL, DEVINI, DRVTBL, |
| Character I/O | |
| | CONST, CONIN, CONOUT, LIST, AUXOUT, AUXIN, LISTST, CONOST, AUXIST, AUXOST |
| Disk I/O | |
| | HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, SECTRN, MULTIO, FLUSH |
| Memory Selects and Moves | |
| | MOVE, XMOVE, SELMEM, SETBNK |
| Clock Support | |
| | TIME |

Table 3-10 is a summary showing the CP/M 3 BIOS function numbers, jump instruction names, and the entry and return parameters of each jump instruction in the table, arranged according to the BIOS function number.

**Table 3-10. CP/M 3 BIOS Function Jump Table Summary**

| No. | Function | Input | Output |
|---|---|---|---|
| 0 | BOOT | None | None |
| 1 | WBOOT | None | None |
| 2 | CONST | None | A=0FFH if ready<br>A=00H if not ready |
| 3 | CONIN | None | A=Con Char |
| 4 | CONOUT | C=Con Char | None |
| 5 | LIST | C=Char | None |
| 6 | AUXOUT | C=Char | None |
| 7 | AUXIN | None | A=Char |
| 8 | HOME | None | None |
| 9 | SELDSK | C=Drive 0-15<br>E=Init Sel Flag | HL=DPH addr<br>HL=000H if invalid dr. |
| 10 | SETTRK | BC=Track No | None |
| 11 | SETSEC | BC=Sector No | None |
| 12 | SETDMA | BC=.DMA | None |
| 13 | READ | None | A=00H if no Err<br>A=01H if Non-recov Err<br>A=0FFH if media changed |
| 14 | WRITE | C=Deblk Code | A=00H if no Err<br>A=01H if Phys Err<br>A=02H if Dsk is R/O<br>A=0FFH if media changed |
| 15 | LISTST | None | A=00H if not ready<br>A=0FFH if ready |
| 16 | SECTRN | BC=Log Sect No<br>DE=Trans Tbl Adr | HL=Phys Sect No |
| 17 | CONOST | None | A=00H if not ready<br>A=0FFH if ready |
| 18 | AUXIST | None | A=00H if not ready<br>A=0FFH if ready |
| 19 | AUXOST | None | A=00H if not ready<br>A=0FFH if ready |
| 20 | DEVTBL | None | HL=Chrtbl addr |
| 21 | DEVINI | C=Dev No 0-15 | None |
| 22 | DRVTBL | None | HL=Drv Tbl addr<br>HL=0FFFFH<br>HL=0FFFEH |
| 23 | MULTIO | C=Mult Sec Cnt | None |
| 24 | FLUSH | None | A=000H if no err<br>A=001H if phys err<br>A=002H if disk R/O |
| 25 | MOVE | HL=Dest Adr<br>DE=Source Adr<br>BC=Count | HL & DE point to next bytes following MOVE |
| 26 | TIME | C=Get/Set Flag | None |
| 27 | SELMEM | A=Mem Bank | None |
| 28 | SETBNK | A=Mem Bank | None |
| 29 | XMOVE | B=Dest Bank<br>C=Source Bank | None |

**Table 3-10. (continued)**

| No. | Function | Input |
|---|---|---|
| 30 | USERF | Reserved for System Implementor |
| 31 | RESERV1 | Reserved for Future Use |
| 32 | RESERV2 | Reserved for Future Use |

### 3.4.1 System Initialization Functions

This section defines the BIOS system initialization routines BOOT, WBOOT, DEVTBL, DEVINI, and DRVTBL.

| BIOS Function 0: BOOT |
|---|
| Get Control from Cold Start Loader and Initialize System |
| Entry Parameters: None<br>Returned Values: None |

The BOOT entry point gets control from the Cold Start Loader in Bank 0 and is responsible for basic system initialization. Any remaining hardware initialization that is not done by the boot ROMs, the Cold Boot Loader, or the LDRBIOS should be performed by the BOOT routine.

The BOOT routine must perform the system initialization outlined in Section 2.3, "System Initialization." This includes initializing Page Zero jumps and loading the CCP. BOOT usually prints a sign-on message, but this can be omitted. Control is then transferred to the CCP in the TPA at 0100H.

To initialize Page Zero, the BOOT routine must place a jump at location 0000H to BIOS_base + 3, the BIOS warm start entry point. The BOOT routine must also place a jump instruction at location 0005H to the address contained in the System Control Block variable, @MXTPA.

The BOOT routine must establish its own stack area if it calls any BDOS or BIOS routines. In a banked system, the stack is in Bank 0 when the Cold BOOT routine is entered. The stack must be placed in common memory.

| BIOS Function 1: WBOOT |
|---|
| Get Control When a Warm Start Occurs |
| Entry Parameters: None<br>Returned Values: None |

The WBOOT entry point is entered when a warm start occurs. A warm start is performed whenever a user program branches to location 0000H or attempts to return to the CCP. The WBOOT routine must perform the system initialization outlined in BIOS Function 0, including initializing Page Zero jumps and loading the CCP.

When your WBOOT routine is complete, it must transfer control to the CCP at location 0100H in the TPA.

Note that the CCP does not reset the disk system at warm start. The CCP resets the disk system when a CTRL-C is pressed following the system prompt.

Note also that the BIOS stack must be in common memory to make BDOS function calls. Only the BOOT and WBOOT routines can perform BDOS function calls.

If the WBOOT routine is reading the CCP from a file, it must set the multisector I/O count, @MLTIO in the System Control Block, to the number of 128-byte records to be read in one operation before reading CCP.COM. You can directly set @MLTIO in the SCB, or you can call BDOS Function 44 to set the multisector count in the SCB.

If blocking/deblocking is done in the BIOS instead of in the BDOS, the WBOOT routine must discard all pending buffers.

| BIOS Function 20: DEVTBL |
|---|
| Return Address of Character I/O Table |
| Entry Parameters: None<br>Returned Values: HL=address of Chrtbl |

The DEVTBL and DEVINI entry points allow you to support device assignment with a flexible, yet completely optional system. It replaces the IOBYTE facility of CP/M 2.2. Note that the CHRTBL must be in common in banked systems.

| BIOS Function 21: DEVINI |
|---|
| Initialize Character I/O Device |
| Entry Parameters: C=device number, 0-15<br>Returned Values: None |

The DEVINI routine initializes the physical character device specified in register C to the baud rate contained in the appropriate entry of the CHRTBL. It need only be supplied if I/O redirection has been implemented and is referenced only by the DEVICE utility supplied with CP/M 3.

| BIOS Function 22: DRVTBL |
|---|
| Return Address of Disk Drive Table |
| Entry Parameters: None<br>Returned Values: HL=Address of Drive Table of Disk Parameter Headers (DPH); Hashing can be utilized if specified by the DPHs referenced by this DRVTBL.<br>HL=0FFFFH if no Drive Table; GENCPM does not set up buffers. Hashing is supported.<br>HL=0FFFEH if no Drive Table; GENCPM does not set up buffers. Hashing is not supported. |

The first instruction of this subroutine must be an LXI H,<address> where <address> is one of the above returned values. The GENCPM utility accesses the address in this instruction to locate the drive table and the disk parameter data structures to determine which system configuration to use.

If you plan to do your own blocking/deblocking, the first instruction of the DRVTBL routine must be the following:

```
lxi     h,0FFFEh
```

You must also set the PSH and PSM fields of the associated Disk Parameter Block to zero.

### 3.4.2 Character I/O Functions

This section defines the CP/M 3 character I/O routines CONST, CONIN, CONOUT, LIST, AUXOUT, AUXIN, LISTST, CONOST, AUXIST, and AUXOST.

CP/M 3 assumes all simple character I/O operations are performed in eight-bit ASCII, upper- and lowercase, with no parity. An ASCII CTRL-Z (1AH) denotes an end-of-file condition for an input device.

In CP/M 3, you can direct each of the five logical character devices to any combination of up to twelve physical devices. Each of the five logical devices has a 16-bit vector in the System Control Block (SCB). Each bit of the vector represents a physical device where bit 15 corresponds to device zero, and bit 4 is device eleven. Bits 0 through 3 are reserved for future system use.

You can use the public names defined in the supplied SCB.ASM file to reference the I/O redirection bit vectors. The names are shown in Table 3-11.

**Table 3-11. I/O Redirection Bit Vectors in SCB**

| Name | Logical Device |
|---|---|
| @CIVEC | Console Input |
| @COVEC | Console Output |
| @AIVEC | Auxiliary Input |
| @AOVEC | Auxiliary Output |
| @LOVEC | List Output |

You should send an output character to all of the devices whose corresponding bit is set. An input character should be read from the first ready device whose corresponding bit is set.

An input status routine should return true if any selected device is ready. An output status routine should return true only if all selected devices are ready.

| BIOS Function 2: CONST | |
|---|---|
| Sample the Status of the Console Input Device | |
| Entry Parameters: | None |
| Returned value: | A=0FFH if a console character is ready to read<br>A=00H if no console character is ready to read |

Read the status of the currently assigned console device and return 0FFH in register A if a character is ready to read, and 00H in register A if no console characters are ready.

| BIOS Function 3: CONIN | |
|---|---|
| Read a Character from the Console | |
| Entry Parameters: | None |
| Returned Values: | A=Console Character |

Read the next console character into register A with no parity. If no console character is ready, wait until a character is available before returning.

| BIOS Function 4: CONOUT | |
|---|---|
| Output Character to Console | |
| Entry Parameters: | C=Console Character |
| Returned Values: | None |

Send the character in register C to the console output device. The character is in ASCII with no parity.

| BIOS Function 5: LIST |
|---|
| Output Character to List Device |
| Entry Parameters: C=Character<br>Returned Values: None |

Send the character from register C to the listing device. The character is in ASCII with no parity.

| BIOS Function 6: AUXOUT |
|---|
| Output a Character to the Auxiliary Output Device |
| Entry Parameters: C=Character<br>Returned Values: None |

Send the character from register C to the currently assigned AUXOUT device. The character is in ASCII with no parity.

| BIOS Function 7: AUXIN |
|---|
| Read a Character from the Auxiliary Input Device |
| Entry Parameters: None<br>Returned Values: A=Character |

Read the next character from the currently assigned AUXIN device into register A with no parity. A returned ASCII CTRL-Z (1AH) reports an end-of-file.

| BIOS Function 15: LISTST | |
|---|---|
| Return the Ready Status of the List Device | |
| Entry Parameters: | None |
| Returned Values: | A=000H if list device is not ready to accept a character<br>A=0FFH if list device is ready to accept a character |

The BIOS LISTST function returns the ready status of the list device.

| BIOS Function 17: CONOST | |
|---|---|
| Return Output Status of Console | |
| Entry Parameters: | None |
| Returned Values: | A=0FFH if ready<br>A=00H if not ready |

The CONOST routine checks the status of the console. CONOST returns an 0FFH if the console is ready to display another character. This entry point allows for full polled handshaking communications support.

| BIOS Function 18: AUXIST | |
|---|---|
| Return Input Status of Auxiliary Port | |
| Entry Parameters: | None |
| Returned Values: | A=0FFH if ready<br>A=000H if not ready |

The AUXIST routine checks the input status of the auxiliary port. This entry point allows full polled handshaking for communications support using an auxiliary port.

| BIOS Function 19: AUXOST | |
|---|---|
| Return Output Status of Auxiliary Port | |
| Entry Parameters: | None |
| Returned Values: | A=0FFH if ready<br>A=000H if not ready |

The AUXOST routine checks the output status of the auxiliary port. This routine allows full polled handshaking for communications support using an auxiliary port.

### 3.4.3 Disk I/O Functions

This section defines the CP/M 3 BIOS disk I/O routines HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, SECTRN, MULTIO, and FLUSH.

| BIOS Function 8: HOME | |
|---|---|
| Select Track 00 of the Specified Drive | |
| Entry Parameters: | None |
| Returned Values: | None |

Return the disk head of the currently selected disk to the track 00 position. Usually, you can translate the HOME call into a call on SETTRK with a parameter of 0.

| BIOS Function 9: SELDSK | |
|---|---|
| **Select the Specified Disk Drive** | |
| Entry Parameters: | C=Disk Drive (0-15)<br>E=Initial Select Flag |
| Returned Values: | HL=Address of Disk Parameter Header (DPH) if drive exists<br>HL=000H if drive does not exist |

Select the disk drive specified in register C for further operations, where register C contains 0 for drive A, 1 for drive B, and so on to 15 for drive P. On each disk select, SELDSK must return in HL the base address of a 25-byte area called the Disk Parameter Header. If there is an attempt to select a nonexistent drive, SELDSK returns HL=0000H as an error indicator.

On entry to SELDSK, you can determine if it is the first time the specified disk is selected. Bit 0, the least significant bit in register E, is set to 0 if the drive has not been previously selected. This information is of interest in systems that read configuration information from the disk to set up a dynamic disk definition table.

When the BDOS calls SELDSK with bit 0 in register E set to 1, SELDSK must return the same Disk Parameter Header address as it returned on the initial call to the drive. SELDSK can only return a 000H indicating an unsuccessful select on the initial select call.

SELDSK must return the address of the Disk Parameter Header on each call. Postpone the actual physical disk select operation until a READ or WRITE is performed, unless I/O is required for automatic density-sensing.

| BIOS Function 10: SETTRK | |
|---|---|
| **Set Specified Track Number** | |
| Entry Parameters: | BC=Track Number |
| Returned Values: | None |

Register BC contains the track number for a subsequent disk access on the currently selected drive. Normally, the track number is saved until the next READ or WRITE occurs.

| BIOS Function 11: SETSEC |
|---|
| Set Specified Sector Number |
| Entry Parameters: BC=Sector Number<br>Returned Values: None |

Register BC contains the sector number for the subsequent disk access on the currently selected drive. This number is the value returned by SECTRN. Usually, you delay actual sector selection until a READ or WRITE operation occurs.

| BIOS Function 12: SETDMA |
|---|
| Set Address for Subsequent Disk I/O |
| Entry Parameters: BC=Direct Memory Access Address<br>Returned Values: None |

Register BC contains the DMA (Direct Memory Access) address for the subsequent READ or WRITE operation. For example, if B = 00H and C = 80H when the BDOS calls SETDMA, then the subsequent read operation reads its data starting at 80H, or the subsequent write operation gets its data from 80H, until the next call to SETDMA occurs.

| BIOS Function 13: READ | |
|---|---|
| Read a Sector from the Specified Drive | |
| Entry Parameters: | None |
| Returned Values: | A=000H if no errors occurred<br>A=001H if nonrecoverable error condition occurred<br>A=0FFH if media has changed |

Assume the BDOS has selected the drive, set the track, set the sector, and specified the DMA address. The READ subroutine attempts to read one sector based upon these parameters, then returns one of the error codes in register A as described above.

If the value in register A is 0, then CP/M 3 assumes that the disk operation completed properly. If an error occurs, the BIOS should attempt several retries to see if the error is recoverable before returning the error code.

If an error occurs in a system that supports automatic density selection, the system should verify the density of the drive. If the density has changed, return a 0FFH in the accumulator. This causes the BDOS to terminate the current operation and relog in the disk.

| BIOS Function 14: WRITE | |
|---|---|
| Write a Sector to the Specified Disk | |
| Entry Parameters: | C=Deblocking Codes |
| Returned Values: | A=000H if no error occurred<br>A=001H if physical error occurred<br>A=002H if disk is Read-Only<br>A=0FFH if media has changed |

Write the data from the currently selected DMA address to the currently selected drive, track, and sector. Upon each call to WRITE, the BDOS provides the following information in register C:

0 = deferred write
1 = nondeferred write
2 = deferred write to the first sector of a new data block

This information is provided for those BIOS implementations that do blocking/deblocking in the BIOS instead of the BDOS.

As in READ, the BIOS should attempt several retries before reporting an error.

If an error occurs in a system that supports automatic density selection, the system should verify the density of the drive. If the density has changed, return a 0FFH in the accumulator. This causes the BDOS to terminate the current operation and relog in the disk.

| BIOS Function 16: SECTRN | |
|---|---|
| Translate Sector Number Given Translate Table | |
| Entry Parameters: | BC=Logical Sector Number<br>DE=Translate Table Address |
| Returned Values: | HL=Physical Sector Number |

SECTRN performs logical sequential sector address to physical sector translation to improve the overall response of CP/M 3. Digital Research ships standard CP/M disk with a skew factor of 6, where six physical sectors are skipped between each logical read operation. This skew factor allows enough time between sectors for most programs on a slow system to process their buffers without missing the next sector. In computer systems that use fast processors, memory, and disk subsystems, you can change the skew factor to improve overall response. Typically, most disk systems perform well with a skew of every other physical sector. You should maintain support of single-density, IBM 3740 compatible disks using a skew factor of 6 in your CP/M 3 system to allow information transfer to and from other CP/M users.

SECTRN receives a logical sector number in BC, and a translate table address in DE. The logical sector number is relative to zero. The translate table address is obtained from the Disk Parameter Block for the currently selected disk. The sector number is used as an index into the translate table, with the resulting physical sector number returned in HL. For standard, single-density, eight-inch disk systems, the tables and indexing code are provided in the sample BIOS and need not be changed.

Certain drive types either do not need skewing or perform the skewing externally from the system software. In this case, the skew table address in the DPH can be set to zero, and the SECTRN routine can check for the zero in DE and return with the physical sector set to the logical sector.

| BIOS Function 23: MULTIO | |
|---|---|
| Set Count of Consecutive Sectors for READ or WRITE | |
| Entry Parameters: | C=Multisector Count |
| Returned Values: | None |

To transfer logically consecutive disk sectors to or from contiguous memory locations, the BDOS issues a MULTIO call, followed by a series of READ or WRITE calls. This allows the BIOS to transfer multiple sectors in a single disk operation. The maximum value of the sector count is dependent on the physical sector size, ranging from 128 with 128-byte sectors, to 4 with 4096-byte sectors. Thus, the BIOS can transfer up to 16K directly to or from the TPA with a single operation.

The BIOS can directly transfer all of the specified sectors to or from the DMA buffer in one operation and then count down the remaining calls to READ or WRITE.

If the disk format uses a skew table to minimize rotational latency when single records are transferred, it is more difficult to optimize transfer time for multisector transfers. One way of utilizing the multisector count with a skewed disk format is to place the sector numbers and associated DMA addresses into a table until either the residual multisector count reaches zero, or the track number changes. Then you can sort the saved requests by physical sector to allow all of the required sectors on the track to be read in one rotation. Each sector must be transferred to or from its proper DMA address.

When an error occurs during a multisector transfer, you can either reset the multiple sector counters in the BIOS and return the error immediately, or you can save the error status and return it to the BDOS on the last READ or WRITE call of the MULTIO operation.

| BIOS Function 24: FLUSH | |
|---|---|
| Force Physical Buffer Flushing for User-supported Deblocking | |
| Entry Parameters: | None |
| Returned Values: | A=000H if no error occurred<br>A=001H if physical error occurred<br>A=002H if disk is Read-Only |

The flush buffers entry point allows the system to force physical *sector buffer* flushing when your BIOS is performing its own record blocking and deblocking.

The BDOS calls the FLUSH routine to ensure that no dirty buffers remain in memory. The BIOS should immediately write any buffers that contain unwritten data.

Normally, the FLUSH function is superfluous, because the BDOS supports blocking/deblocking internally. It is required, however, for those systems that support blocking/deblocking in the BIOS, as many CP/M 2.2 systems do.

**Note:** *if you do not* implement FLUSH, the routine must return a zero in register A. You can accomplish this with the following instructions:

```
        xra     a
        ret
```

### 3.4.4 Memory Select and Move Functions

This section defines the memory management functions MOVE, XMOVE, SELMEM, and SETBNK.

| BIOS Function 25: MOVE | |
|---|---|
| Memory-to-Memory Block Move | |
| Entry Parameters: | HL=Destination address<br>DE=Source address<br>BC=Count |
| Returned Values: | HL and DE must point to next bytes following move operation |

The BDOS calls the MOVE routine to perform memory to memory block moves to allow use of the Z80 LDIR instruction or special DMA hardware, if available. Note that the arguments in HL and DE are reversed from the Z80 machine instruction, necessitating the use of XCHG instructions on either side of the LDIR. The BDOS uses this routine for all large memory copy operations. On return, the HL and DE registers are expected to point to the next bytes following the move.

Usually, the BDOS expects MOVE to transfer data within the currently selected bank or common memory. However, if the BDOS calls the XMOVE entry point before calling MOVE, the MOVE routine must perform an interbank transfer.

| BIOS Function 27: SELMEM | |
|---|---|
| Select Memory Bank | |
| Entry Parameters: | A=Memory Bank |
| Returned Values: | None |

The SELMEM entry point is only present in banked systems. The banked version of the CP/M 3 BDOS calls SELMEM to select the current memory bank for further instruction execution or buffer references. You must preserve or restore all registers other than the accumulator, A, upon exit.

| BIOS Function 28: SETBNK | |
|---|---|
| Specify Bank for DMA Operation | |
| Entry Parameters: | A=Memory Bank |
| Returned Values: | None |

SETBNK only occurs in the banked version of CP/M 3. SETBNK specifies the bank that the subsequent disk READ or WRITE routine must use for memory transfers. The BDOS always makes a call to SETBNK to identify the DMA bank before performing a READ or WRITE call. Note that the BDOS does not reference banks other than 0 or 1 unless another bank is specified by the BANK field of a Data Buffer Control Block (BCB).

| BIOS Function 29: XMOVE | |
|---|---|
| Set Banks for Following MOVE | |
| Entry Parameters: | B=destination bank<br>C=source bank |
| Returned Values: | None |

XMOVE is provided for banked systems that support memory-to-memory DMA transfers over the entire extended address range. Systems with this feature can have their data buffers located in an

alternate bank instead of in common memory, as is usually required. An XMOVE call affects only the following MOVE call. All subsequent MOVE calls apply to the memory selected by the latest call to SELMEM. After a call to the XMOVE function, the following call to the MOVE function is not more than 128 bytes of data. If you do not implement XMOVE, the first instruction must be a RET instruction.

### 3.4.5 Clock Support Function

This section defines the clock support function TIME.

| BIOS Function 26: TIME | |
|---|---|
| Get and Set Time | |
| Entry Parameters: | C=Time Get/Set Flag |
| Returned values: | None |

The BDOS calls the TIME function to indicate to the BIOS whether it has just set the Time and Date fields in the SCB, or whether the BDOS is about to get the Time and Date from the SCB. On entry to the TIME function, a zero in register C indicates that the BIOS should update the Time and Date fields in the SCB. A 0FFH in register C indicates that the BDOS has just set the Time and Date in the SCB and the BIOS should update its clock. Upon exit, you must restore register pairs HL and DE to their entry values.

This entry point is for systems that must interrogate the clock to determine the time. Systems in which the clock is capable of generating an interrupt should use an interrupt service routine to set the Time and Date fields on a regular basis.

## 3.5 Banking Considerations

This section discusses considerations for separating your BIOS into resident and banked modules. You can place part of your customized BIOS in common memory, and part of it in Bank 0. However, the following data structures and routines must remain in common memory:

- the BIOS stack
- the BIOS jump vector
- Disk Parameter Blocks
- memory management routines
- the CHRTBL data structure
- all character I/O routines
- portions of the disk I/O routines

You can place portions of the disk I/O routines in the system bank, Bank 0. In a banked environment, if the disk I/O hardware supports DMA transfers to and from banks other than the currently selected bank, the disk I/O drivers can reside in Bank 0. If the system has a DMA controller that supports block moves from memory to memory between banks, CP/M 3 also allows you to place the blocking and deblocking buffers in any bank other than Bank 1, instead of common memory.

If your disk controller supports data transfers only into the currently selected bank, then the code that initiates and performs a data transfer must reside in common memory. In this case, the disk I/O transfer routines must select the DMA bank, perform the transfer, then reselect Bank 0. The routine in common memory performs the following procedure:

1) Selects the DMA bank that SETBNK saved.
2) Performs physical I/O.
3) Reselects Bank 0.
4) Returns to the calling READ or WRITE routine in Bank 0.

Note that Bank 0 is in context (selected) when the BDOS calls the system initialization functions BOOT and DRVTBL; the disk I/O routines HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, SECTRN, MULTIO, and FLUSH; and the memory management routines XMOVE and SETBNK.

Bank 0 or Bank 1 is in context when the BDOS calls the system initialization routines WBOOT, DEVTBL, and DEVINI; the character I/O routines CONST, CONIN, CONOUT, LIST, AUXOUT, AUXIN, LISTST, CONOST, AUXIST, and AUXOST, the memory select and move routines MOVE and SELMEM, and the clock support routine TIME.

You can place a portion of the character I/O routines in Bank 0 if you place the following procedure in common memory.

1) Swap stacks to a local stack in common.
2) Save the current bank.
3) Select Bank 0.
4) Call the appropriate character I/O routine.
5) Reselect the saved bank.
6) Restore the stack.

### 3.6 Assembling and Linking Your BIOS

This section assumes you have developed a BIOS3.ASM or BNKBIOS3.ASM file appropriate to your specific hardware environment. Use the Digital Research Relocatable Macro Assembler RMAC™ to assemble the BIOS. Use the Digital Research Linker LINK-80™ to create the BIOS3.SPR and BNKBIOS3.SPR files. The SPR files are part of the input to the GENCPM program.

In a banked environment, your CP/M 3 BIOS can consist of two segments: a banked segment and a common segment. This allows you to minimize common memory usage to maximize the size of the TPA. To prepare a banked BIOS, place code and data that must reside in common in the CSEG segment, and code and data that can reside in the system bank in the DSEG segment. When you link the BIOS, LINK-80 creates the BNKBIOS3.SPR file with all the CSEG code and data first, then the DSEG code and data.

After assembling the BIOS with RMAC, link your BNKBIOS using LINK-80 with the [B] option. The [B] option aligns the DSEG on a page boundary, and places the length of the CSEG into the BNKBIOS3.SPR header page.

Use the following procedure to prepare a BIOS3.SPR or BNKBIOS3.SPR file from your customized BIOS.

1) Assemble your BIOS3.ASM or BNKBIOS3.ASM file with the relocatable assembler RMAC.COM to produce a relocatable file of type REL. Assemble SCB.ASM to produce the relocatable file SCB.REL.

   Assembling the Nonbanked BIOS:

   A>**RMAC BIOS3**

   Assembling the Banked BIOS:

   A>**RMAC BNKBIOS3**

2) Link the BIOS3.REL or BNKBIOS3.REL file and the SCB.REL file with LINK-80 to produce the BIOS3.SPR or BNKBIOS3.SPR file. The [OS] option with LINK causes the output of a System Page Relocatable (SPR) file.

   Linking the Nonbanked BIOS:

   A>**LINK BIOS3[OS]=BIOS3,SCB**

   Linking the Banked BIOS:

   A>**LINK BNKBIOS3[B]=BNKBIOS3,SCB**

The preceding examples show command lines for linking a banked and nonbanked BIOS. In these examples, the BIOS3.REL and BNKBIOS3.REL are the files of your assembled BIOS. SCB.REL contains the definitions of the System Control Block variables. The [B] option implies the [OS] option.

End of Section 3

# Section 4
# CP/M 3 Sample BIOS Modules

This section discusses the modular organization of the example CP/M 3 BIOS on your distribution disk. For previous CP/M operating systems, it was necessary to generate all input/output drivers from a single assembler source file. Such a file is difficult to maintain when the BIOS supports several peripherals. As a result, Digital Research is distributing the BIOS for CP/M 3 in several small modules.

The organization of the BIOS into separate modules allows you to write or modify any I/O driver independently of the other modules. For example, you can easily add another disk I/O driver for a new controller with minimum impact on the other parts of the BIOS.

## 4.1 Functional Summary of BIOS Modules

The modules of the BIOS are BIOSKRNL.ASM, SCB.ASM, BOOT.ASM, MOVE.ASM, CHARIO.ASM, DRVTBL.ASM, and a disk I/O module for each supported disk controller in the configuration.

BIOSKRNL.ASM is the kernel, root, or supervisor module of the BIOS. The SCB.ASM module contains references to locations in the System Control Block. You can customize the other modules to support any hardware configuration. To customize your system, add or modify external modules other than the kernel and the SCB.ASM module.

Digital Research supplies the BIOSKRNL.ASM module. This module is the fixed, invariant portion of the BIOS, and the interface from the BDOS to all BIOS functions. It is supplied in source form for reference only, and you should not modify it except for the equate statement described in the following paragraph.

You must be sure the equate statement (banked equ true) at the start of the BIOSKRNL.ASM source file is correct for your system configuration. Digital Research distributes the BIOSKRNL.ASM file for a banked system. If you are creating a BIOS for a nonbanked system, change the equate statement to the following:

```
banked equ false
```

and reassemble with RMAC. This is the only change you should make to the BIOSKRNL.ASM file.

Table 4-1 summarizes the modules in the CP/M 3 BIOS.

**Table 4-1. CP/M 3 BIOS Module Function Summary**

| Module | Function |
|---|---|
| BIOSKRNL.ASM | Performs basic system initialization, and dispatches character and disk I/O. |
| SCB.ASM module | Contains the public definitions of the various fields in the System Control Block. The BIOS can reference the public variables. |
| BOOT.ASM module | Performs system initialization other than character and disk I/O. BOOT loads the CCP for cold starts and reloads it for warm starts. |
| CHARIO.ASM module | Performs all character device initialization, input, output, and status polling. CHARIO contains the character device characteristics table. |
| DRVTBL.ASM module | Points to the data structures for each configured disk drive. The drive table determines which physical disk unit is associated with which logical drive. The data structure for each disk drive is called an Extended Disk Parameter Header (XDPH). |
| Disk I/O modules | Initialize disk controllers and execute READ and WRITE code for disk controllers. You must provide an XDPH for each supported unit, and a separate disk I/O module for each controller in the system. To add another disk controller for which a prewritten module exists, add its XDPH names to the DRVTBL and link in the new module. |

**Table 4-1. (continued)**

| Module | Function |
|---|---|
| MOVE.ASM module | Performs memory-to-memory moves and bank selects. |

## 4.2 Conventions Used in BIOS Modules

The Digital Research RMAC relocating assembler and LINK-80 linkage editor allow a module to reference a symbol contained in another module by name. This is called an external reference. The Microsoft® relocatable object module format that RMAC and LINK use allows six-character names for externally defined symbols. External names must be declared PUBLIC in the module in which they are defined. The external names must be declared EXTRN in any modules that reference them.

The modular BIOS defines a number of external names for specific purposes. Some of these are defined as public in the root module, BIOSKRNL.ASM. Others are declared external in the root and must be defined by the system implementor. Section 4.4 contains a table summarizing all predefined external symbols used by the modular BIOS.

External names can refer to either code or data. All predefined external names in the modular BIOS prefixed with a @ character refer to data items. All external names prefixed with a ? character refer to a code label. To prevent conflicts with future extensions, user-defined external names should not contain these characters.

## 4.3 Interactions of Modules

The root module of the BIOS, BIOSKRNL.ASM, handles all BDOS calls, performs interfacing functions, and simplifies the individual modules you need to create.

### 4.3.1 Initial Boot

BIOSKRNL.ASM initializes all configured devices in the following order:

1) BIOSKRNL calls ?CINIT in the CHARIO module for each of the 16 character devices and initializes the devices.

2) BIOSKRNL invokes the INIT entry point of each XDPH in the FD1797SD module.

3) BIOSKRNL calls the ?INIT entry of the BOOT module to initialize other system hardware, such as memory controllers, interrupts, and clocks. It prints a sign-on message specific to the system, if desired.

4) BIOSKRNL calls ?LDCCP in the BOOT module to load the CCP into the TPA.

5) The BIOSKRNL module sets up Page Zero of the TPA with the appropriate jump vectors, and passes control to the CCP.

### 4.3.2 Character I/O Operation

The CHARIO module performs all physical character I/O. This module contains both the character device table (@CTBL) and the routines for character input, output, initialization, and status polling. The character device table, @CTBL, contains the ASCII name of each device, mode information, and the current baud rate of serial devices.

To support logical to physical redirection of character devices, CP/M 3 supplies a 16-bit assignment vector for each logical device. The bits in these vectors correspond to the physical devices. The character I/O interface routines in BIOSKRNL handle all device assignment, calling the appropriate character I/O routines with the correct device number. The BIOSKRNL module also handles XON/XOFF processing on output devices where it is enabled.

You can use the DEVICE utility to assign several physical devices to a logical device. The BIOSKRNL root module polls the assigned physical devices, and either reads a character from the first ready input device that is selected, or sends the character to all of the selected output devices as they become ready.

### 4.3.3 Disk I/O Operation

The BIOSKRNL module handles all BIOS calls associated with disk I/O. It initializes global variables with the parameters for each operation, then invokes the READ or WRITE routine for a particular controller. The SELDSK routine in the BIOSKRNL calls the LOGIN routine for a controller when the BDOS initiates a drive login. This allows disk density or media type to be automatically determined.

The DRVTBL module contains the sixteen-word drive table, @DTBL. The order of the entries in @DTBL determines the logical to physical drive assignment. Each word in @DTBL contains the address of a DPH, which is part of an XDPH, as shown in Table 4-10. The word contains a zero if the drive does not exist. The XDPH contains the addresses of the INIT, LOGIN, READ, and WRITE entry points of the I/O driver for a particular controller. When the actual drivers are called, globally accessible variables contain the various parameters of the operation, such as the track and sector.

## 4.4 Predefined Variables and Subroutines

The modules of the BIOS define public variables which other modules can reference. Table 4-2 contains a summary of each public symbol and the module that defines it.

**Table 4-2. Public Symbols in CP/M 3 BIOS**

| Symbol | Function and Use | Defined in Module |
|---|---|---|
| @ADRV | Byte, Absolute drive code | BIOSKRNL |
| @CBNK | Byte, Current CPU bank | BIOSKRNL |
| @CNT | Byte, Multisector count | BIOSKRNL |
| @CTBL | Table, Character device table | CHARIO |
| @DBNK | Byte, Bank for disk I/O | BIOSKRNL |
| @DMA | Word, DMA address | BIOSKRNL |
| @DTBL | Table, Drive table | DRVTBL |
| @RDRV | Byte, Relative drive code (UNIT) | BIOSKRNL |
| @SECT | Word, Sector address | BIOSKRNL |
| @TRK | Word, Track number | BIOSKRNL |
| ?BANK | Bank select | MOVE |
| ?CI | Character device input | CHARIO |
| ?CINIT | Character device initialization | CHARIO |
| ?CIST | Character device input status | CHARIO |
| ?CO | Character device output | CHARIO |
| ?COST | Character device output status | CHARIO |
| ?INIT | General initialization | BOOT |
| ?LDCCP | Load CCP for cold start | BOOT |
| ?MOVE | Move memory to memory | MOVE |
| ?PDEC | Print decimal number | BIOSKRNL |
| ?PDERR | Print BIOS disk error header | BIOSKRNL |
| ?PMSG | Print message | BIOSKRNL |
| ?RLCCP | Reload CCP for warm start | BOOT |
| ?XMOVE | Set banks for extended move | MOVE |
| ?TIME | Set or Get time | BOOT |

The System Control Block defines public variables that other modules can reference. The System Control Block variables @CIVEC, @COVEC, @AIVEC, @AOVEC, and @LOVEC are referenced by BIOSKRNL.ASM. The variable @BNKBF can be used by ?LDCCP and ?RLCCP to implement interbank block moves. The public variable names @ERMDE, @FX, @RESEL, @VINFO, @CRDSK, @USRCD, and @CRDMA are used for error routines which intercept BDOS errors. The publics @DATE, @HOUR, @MIN, and @SEC can be updated by an interrupt-driven real-time clock. @MXTPA contains the current BDOS entry point.

Disk I/O operation parameters are passed in the following global variables, as shown in Table 4-3.

**Table 4-3. Global Variables in BIOSKRNL.ASM**

| Variable | Meaning |
|---|---|
| @ADRV | Byte; contains the absolute drive code (0 through F for A through P) that CP/M is referencing for READ and WRITE operations. The SELDSK routine in the BIOSKRNL module obtains this value from the BDOS and places it in @DRV. The absolute drive code is used to print error messages. |
| @RDRV | Byte; contains the relative drive code for READ and WRITE operations. The relative drive code is the UNIT number of the controller in a given disk I/O module. BIOSKRNL obtains the unit number from the XDPH. This is the actual drive code a driver should send to the controller. |
| @TRK | Word; contains the starting track for READ and WRITE. |
| @SECT | Word; contains the starting sector for READ and WRITE. |
| @DMA | Word; contains the starting disk transfer address. |
| @DBNK | Byte; contains the bank of the DMA buffer. |
| @CNT | Byte; contains the physical sector count for the operations that follow. |
| @CBNK | Byte; contains the current bank for code execution. |

Several utility subroutines are defined in the BIOSKRNL.ASM module, as shown in Table 4-4.

**Table 4-4. Public Utility Subroutines in BIOSKRNL.ASM**

| Utility | Meaning |
|---|---|
| ?PMSG | Print string starting at <HL>, stop at null (0). |
| ?PDEC | Print binary number in decimal from HL. |
| ?PDERR | Print disk error message header using current disk parameters: <CR><LF>BIOS Error on d:, T-nn, S-nn. |

All BIOS entry points in the jump vector are declared as public for general reference by other BIOS modules, as shown in Table 4-5.

**Table 4-5. Public Names in the BIOS Jump Vector**

| Public Name | Function |
|---|---|
| ?BOOT | Cold boot entry |
| ?WBOOT | Warm boot entry |
| ?CONST | Console input status |
| ?CONIN | Console input |
| ?CONO | Console output |
| ?LIST | List output |
| ?AUXO | Auxiliary output |
| ?AUXI | Auxiliary input |
| ?HOME | Home disk drive |
| ?SLDSK | Select disk drive |
| ?STTRK | Set track |
| ?STSEC | Set sector |
| ?STDMA | Set DMA address |
| ?READ | Read record |
| ?WRITE | Write record |
| ?LISTS | List status |
| ?SCTRN | Translate sector |
| ?CONOS | Console output status |
| ?AUXIS | Auxiliary input status |
| ?AUXOS | Auxiliary output status |
| ?DVTBL | Return character device table address |
| ?DEVIN | Initialize character device |
| ?DRTBL | Return disk drive table address |
| ?MLTIO | Set multiple sector count |
| ?FLUSH | Flush deblocking buffers (not implemented) |
| ?MOV | Move memory block |
| ?TIM | Signal set or get time from clock |
| ?BNKSL | Set bank for further execution |
| ?STBNK | Set bank for DMA |
| ?XMOV | Set banks for next move |

## 4.5 BOOT Module

The BOOT module performs general system initialization, and loads and reloads the CCP. Table 4-6 shows the entry points of the BOOT module.

**Table 4-6. BOOT Module Entry Points**

| Module | Meaning |
|---|---|
| ?INIT | The BIOSKRNL module calls ?INIT during cold start to perform hardware initialization other than character and disk I/O. Typically, this hardware can include time-of-day clocks, interrupt systems, and special I/O ports used for bank selection. |
| ?LDCCP | BIOSKRNL calls ?LDCCP during cold start to load the CCP into the TPA. The CCP can be loaded either from the system tracks of the boot device or from a file, at the discretion of the system implementor. In a banked system, you can place a copy of the CCP in a reserved area of another bank to increase the performance of the ?RLCCP routine. |
| ?RLCCP | BIOSKRNL calls ?RLCCP during warm start to reload the CCP into the TPA. In a banked system, the CCP can be copied from an alternate bank to eliminate any disk access. Otherwise, the CCP should be loaded from either the system tracks of the boot device or from a file. |

## 4.6 Character I/O

The CHARIO module handles all character device interfacing. The CHARIO module contains the character device definition table @CTBL, the character input routine ?CI, the character output routine ?CO, the character input status routine ?CIST, the character output status routine ?COST, and the character device initialization routine ?CINIT.

The BIOS root module, BIOSKRNL.ASM, handles all character I/O redirection. This module determines the appropriate devices to perform operations and executes the actual operation by calling ?CI, ?CO, ?CIST, and ?COST with the proper device number(s).

@CTBL is the external name for the structure CHRTBL described in Section 3 of this manual. @CTBL contains an 8-byte entry for each physical device defined by this BIOS. The table is terminated by a zero byte after the last entry.

The first field of the character device table, @CTBL, is the 6-byte device name. This device name should be all upper-case, left-justified, and padded with ASCII spaces (20H).

The second field of @CTBL is 1 byte containing bits that indicate the type of device and its current mode, as shown in Table 4-7.

**Table 4-7. Mode Bits**

| Mode Bits | Meaning |
|---|---|
| 00000001 | Input device (such as a keyboard) |
| 00000010 | Output device (such as a printer) |
| 00000011 | Input/output device (such as a terminal or modem) |
| 00000100 | Device has software-selectable baud rates |
| 00001000 | Device may use XON protocol |
| 00010000 | XON/XOFF protocol enabled |

The third field of @CTBL is 1 byte and contains the current baud rate for serial devices. The high-order nibble of this field is reserved for future use and should be set to zero. The low-order four bits contain the current baud rate as shown in Table 4-8. Many systems do not support all of these baud rates.

**Table 4-8. Baud Rates for Serial Devices**

| Decimal | Binary | Baud Rate |
|---|---|---|
| 0 | 0000 | none |
| 1 | 0001 | 50 |
| 2 | 0010 | 75 |
| 3 | 0011 | 110 |
| 4 | 0100 | 134.5 |
| 5 | 0101 | 150 |
| 6 | 0110 | 300 |
| 7 | 0111 | 600 |
| 8 | 1000 | 1200 |
| 9 | 1001 | 1800 |
| 10 | 1010 | 2400 |
| 11 | 1011 | 3600 |
| 12 | 1100 | 4800 |
| 13 | 1101 | 7200 |
| 14 | 1110 | 9600 |
| 15 | 1111 | 19200 |

Table 4-9 shows the entry points to the routines in the CHARIO module. The BIOSKRNL module calls these routines to perform machine-dependent character I/O.

**Table 4-9. Character Device Labels**

| Label | Meaning |
|---|---|
| ?CI | Character Device Input<br><br>?CI is called with a device number in register B. It should wait for the next available input character, then return the character in register A. The character should be in 8-bit ASCII with no parity. |
| ?CO | Character Device Output<br><br>?CO is called with a device number in register B and a character in register C. It should wait until the device is ready to accept another character and then send the character. The character is in 8-bit ASCII with no parity. |
| ?CIST | Character Device Input Status<br><br>?CIST is called with a device number in register B. It should return with register A set to zero if the device specified has no input character ready; and should return with A set to 0FFH if the device specified has an input character ready to be read. |
| ?COST | Character Device Output Status<br><br>?COST is called with a device number in register B. It should return with register A set to zero if the device specified cannot accept a character immediately, and should return with A set to 0FFH if the device is ready to accept a character. |
| ?CINIT | Character Device Initialization<br><br>?CINIT is called for each of the 16 character devices, and initializes the devices. Register C contains the device number. The ?CINIT routine initializes the physical character device specified in register C to the baud rate contained in the appropriate entry of the CHRTBL. You only need to supply this routine if I/O redirection has been implemented. It is referenced only by the DEVICE utility supplied with CP/M 3. |

## 4.7 Disk I/O

The separation of the disk I/O section of the BIOS into several modules allows you to support each particular disk controller independently from the rest of the system. A manufacturer can supply the code for a controller in object module form, and you can link it into any existing modular BIOS to function with other controllers in the system.

The data structure called the Extended Disk Parameter Header, or XDPH, contains all the necessary information about a disk drive. BIOSKRNL.ASM locates the XDPH for a particular logical drive using the Drive Table. The XDPH contains the addresses of the READ, WRITE, initialization, and login routines. The XDPH also contains the relative unit number of the drive on the controller, the current media type, and the Disk Parameter Header (DPH) that the BDOS requires. Section 3 of this manual describes the Disk Parameter Header.

The code to read and write from a particular drive is independent of the actual CP/M logical drive assignment, and works with the relative unit number of the drive on the controller. The position of the XDPH entry in the DRVTBL determines the actual CP/M 3 drive code.

### 4.7.1 Disk I/O Structure

The BIOS requires a DRVTBL module to locate the disk driver. It also requires a disk module for each controller that is supported.

The drive table module, DRVTBL, contains the addresses of each XDPH defined in the system. Each XDPH referenced in the DRVTBL must be declared external to link the table with the actual disk modules.

The XDPHs are the only public entry points in the disk I/O modules. The root module references the XDPHs to locate the actual I/O driver code to perform sector READS and WRITES. When the READ and WRITE routines are called, the parameters controlling the READ or WRITE operation are contained in a series of global variables that are declared public in the root module.

### 4.7.2 Drive Table Module (DRVTBL)

The drive table module, DRVTBL, defines the CP/M absolute drive codes associated with the physical disks.

The DRVTBL module contains one public label, @DTBL. @DTBL is a 16-word table containing the addresses of up to 16 XDPH's. Each XDPH name must be declared external in the DRVTBL. The first entry corresponds to drive A, and the last to drive P. You must set an entry to 0 if the corresponding drive is undefined. Selecting an undefined drive causes a BDOS SELECT error.

### 4.7.3 Extended Disk Parameter Headers (XDPHs)

An Extended Disk Parameter Header (XDPH) consists of a prefix and a regular Disk Parameter Header as described in Section 3. The label of a XDPH references the start of the DPH. The fields of the prefix are located at relative offsets from the XDPH label.

The XDPHs for each unit of a controller are the only entry points in a particular disk drive module. They contain both the DPH for the drive and the addresses of the various action routines for that drive, including READ, WRITE, and initialization. Figure 4-1 shows the format of the Extended Disk Parameter Header.

| ADDRESS | LOW BYTE (0–7) | HIGH BYTE (8–15) | |
|---|---|---|---|
| XDPH-10 | addr of sector WRITE | | |
| XDPH-8 | addr of sector READ | | |
| XDPH-6 | addr of drive LOGIN | | |
| XDPH-4 | addr of drive INIT | | |
| XDPH-2 | unit | type | |
| XDPH+0 | addr of translate table | | ← start of regular DPH |
| XDPH+2 | 0 | 0 | |
| XDPH+4 | 0 | 0 | |
| XDPH+6 | 0 | 0 | |
| XDPH+8 | 0 | 0 | |
| XDPH+10 | Media Flag | 0 | |
| XDPH+12 | addr of DPB | | |
| XDPH+14 | addr of CSV | | |
| XDPH+16 | addr of ALV | | |
| XDPH+18 | addr of DIRBCB | | |
| XDPH+20 | addr of DTABCB | | |
| XDPH+22 | addr of HASH | | |
| XDPH+24 | hash bank | | |

**Figure 4-1. XDPH Format**

Table 4-10 describes the fields of each Extended Disk Parameter Header.

**Table 4-10. Fields of Each XDPH**

| Field | Meaning |
|---|---|
| WRITE | The WRITE word contains the address of the sector WRITE routine for the drive. |
| READ | The READ word contains the address of the sector READ routine for the drive. |
| LOGIN | The LOGIN word contains the address of the LOGIN routine for the drive. |
| INIT | The INIT word contains the address of the first-time initialization code for the drive. |
| UNIT | The UNIT byte contains the drive code relative to the disk controller. This is the value placed in @RDRV prior to calling the READ, WRITE, and LOGIN entry points of the drive. |
| TYPE | The TYPE byte is unused by the BIOS root, and is reserved for the driver to keep the current density or media type to support multiple-format disk subsystems. |
| regular DPH | The remaining fields of the XDPH comprise a standard DPH, as discussed in Section 3 of this manual. |

### 4.7.4 Subroutine Entry Points

The pointers contained in the XDPH reference the actual code entry points to a disk driver module. These routines are not declared public. Only the XDPH itself is public. The BIOS root references the XDPHs only through the @DTBL. Table 4-11 shows the BIOS subroutine entry points.

**Table 4-11. Subroutine Entry Points**

| Entry Point | Meaning |
|---|---|
| WRITE | When the WRITE routine is called, the address of the XDPH is passed in registers DE. The parameters for the WRITE operation are contained in the public variables @ADRV, @RDRV, @TRK, @SECT, @DMA, and @DBNK. The WRITE routine should return an error code in register A. The code 00 means a successful operation, 01 means a permanent error occurred, and 02 means the drive is write-protected if that feature is supported. |
| READ | When the READ routine is called, the address of the XDPH is contained in registers DE. The parameters for the READ operation are contained in the public variables @ADRV, @RDRV, @TRK, @SECT, @DMA, and @DBNK. The READ routine should return an error code in register A. A code of 00 means a successful operation and 01 means a permanent error occurred. |
| LOGIN | The LOGIN routine is called before the BDOS logs into the drive, and allows the automatic determination of density. The LOGIN routine can alter the various parameters in the DPH, including the translate table address (TRANS) and the Disk Parameter Block (DPB). The LOGIN routine can also set the TYPE byte. On single media type systems, the LOGIN routine can simply return. When LOGIN is called, the registers DE point to the XDPH for this drive. |
| INIT | The BOOT entry of the BIOSKRNL module calls each INIT routine during cold start and prior to any other disk accesses. INIT can perform any necessary hardware initialization, such as setting up the controller and interrupt vectors, if any. |

### 4.7.5 Error Handling and Recovery

The READ and WRITE routines should perform several retries of an operation that produces an error. If the error is related to a seek operation or a record not found condition, the retry routine can home or restore the drive, and then seek the correct track. The exact sequence of events is hardware-dependent.

When a nonrecoverable error occurs, the READ or WRITE routines can print an error message informing the operator of the details of the error. The BIOSKRNL module supplies a subroutine, ?PDERR, to print a standard BIOS error message header. This routine prints the following message:

```
BIOS Err on D: T-nn S-nn
```

The D: is the selected drive, and T-nn and S-nn display the track and sector number for the operation. The READ and WRITE routines should print the exact cause of the error after this message, such as Not Ready, or Write Protect. The driver can then ask the operator if additional retries are desired, and return an error code to the BDOS if they are not.

However, if the @ERMDE byte in the System Control Block indicates the BDOS is returning error codes to the application program without printing error messages, the BIOS should simply return an error without any message.

### 4.7.6 Multiple Sector I/O

The root module global variable @CNT contains the multisector count. Refer to Sections 2.5 and 3.4.3 for a discussion of the considerations regarding multirecord I/O.

## 4.8 MOVE Module

The MOVE Module performs memory-to-memory block moves and controls bank selection. The ?MOVE and ?XMOVE entry points correspond directly to the MOVE and XMOVE jump vector routines documented in Section 3. Table 4-12 shows the entry points for the MOVE module.

**Table 4-12. Move Module Entry Points**

| Entry Point | Meaning |
|---|---|
| ?MOVE | Memory-to-memory move<br><br>?MOVE is called with the source address for the move in register DE, the destination address in register HL, and the byte count in register BC. If ?XMOVE has been called since the last call to ?MOVE, an interbank move must be performed. On return, registers HL and DE must point to the next bytes after the MOVE. This routine can use special DMA hardware for the interbank move capability, and can use the Z80 LDIR instruction for intrabank moves. |
| ?XMOVE | Set banks for one following ?MOVE<br><br>?XMOVE is called with the destination bank in register B and the source bank in register C. Interbank moves are only invoked if the DPHs specify deblocking buffers in alternate banks. ?XMOVE only applies to one call to ?MOVE. (Not implemented in the example.) |
| ?BANK | Set bank for execution<br><br>?BANK is called with the bank address in register A. This bank address has already been stored in @CBNK for future reference. All registers except A must be maintained upon return. |

### 4.9 Linking Modules into the BIOS

The following lines are examples of typical link commands to build a modular BIOS ready for system generation with GENCPM:

```
LINK BNKBIOS3[b]=BNKBIOS,SCB,BOOT,CHARIO,MOVE,DRVTBL,<disk_modules>

LINK BIOS3[os]=BIOS,SCB,BOOT,CHARIO,MOVE,DRVTBL,<disk_modules>
```

End of Section 4

# Section 5
# System Generation

This section describes the use of the GENCPM utility to create a memory image CPM3.SYS file containing the elements of the CP/M 3 operating system. This section also describes customizing the LDRBIOS portion of the CPMLDR program, and the operation of CPMLDR to read the CPM3.SYS file into memory. Finally, this section describes the procedure to follow to boot CP/M 3.

In the nonbanked system, GENCPM creates the CPM3.SYS file from the BDOS3.SPR and your customized BIOS3.SPR files. In the banked system, GENCPM creates the CPM3.SYS file from the RESBDOS3.SPR file, the BNKBDOS3.SPR file, and your customized BNKBIOS3.SPR file.

If your BIOS contains a segment that can reside in banked memory, GENCPM separates the code and data in BNKBIOS3.SPR into a banked portion which resides in Bank 0 just below common memory, and a resident portion which resides in common memory.

GENCPM relocates the system modules, and can allocate physical record buffers, allocation vectors, checksum vectors, and hash tables as requested in the BIOS data structures. It also relocates references to the System Control Block, as described on page 27. GENCPM accepts its command input from a file, GENCPM.DAT, or interactively from the console.

## 5.1 GENCPM Utility

Syntax:

GENCPM {AUTO | AUTO DISPLAY}

Purpose:

GENCPM creates a memory image CPM3.SYS file, containing the CP/M 3 BDOS and customized BIOS. The GENCPM utility performs late resolution of intermodule references between system modules. GENCPM can accept its command input interactively from the console or from a file GENCPM.DAT.

In the nonbanked system, GENCPM creates a CPM3.SYS file from the BDOS3.SPR and BIOS3.SPR files. In the banked system, GENCPM creates the CPM3.SYS file from the RESBDOS3.SPR, the BNKBDOS3.SPR and the BNKBIOS3.SPR files. Remember to back up your CPM3.SYS file before executing GENCPM, because GENCPM deletes any existing CPM3.SYS file before it generates a new system.

Input Files:

| Banked System | Nonbanked System |
|---|---|
| BNKBIOS3.SPR | BIOS3.SPR |
| RESBDOS3.SPR | BDOS3.SPR |
| BNKBDOS3.SPR | |

Optionally GENCPM.DAT

Output File:

CPM3.SYS

Optionally GENCPM.DAT

GENCPM determines the location of the system modules in memory and, optionally, the number of physical record buffers allocated to the system. GENCPM can specify the location of hash tables requested by the Disk Parameter Headers (DPHs) in the BIOS. GENCPM can allocate all required disk buffer space and create all the required Buffer Control Blocks (BCBs). GENCPM can also create checksum vectors and allocation vectors.

GENCPM can get its input from a file GENCPM.DAT. The values in the file replace the default values of GENCPM. If you enter the AUTO parameter in the command line GENCPM gets its input from the file GENCPM.DAT and generates a new system displaying only its sign-on and sign-off messages on the console. If AUTO is specified and a GENCPM.DAT file does not exist on the current drive, GENCPM reverts to manual generation.

If you enter the AUTO DISPLAY parameter in the command line, GENCPM automatically generates a new system and displays all questions on the console. If AUTO DISPLAY is specified and a GENCPM.DAT file does not exist on the current drive, GENCPM reverts to manual generation. If GENCPM is running in AUTO mode and an error occurs, it reverts to manual generation and starts from the beginning.

The GENCPM.DAT file is an ASCII file of variable names and their associated values. In the following discussion, a variable name in the GENCPM.DAT file is referred to as a Question Variable. A line in the GENCPM.DAT file takes the following general form:

```
Question Variable = value | ? | ?value <CR><LF>

      value = #decimal value
              or hexadecimal value
              or drive letter (A - P)
              or Yes, No, Y, or N
```

You can specify a default value by following a question mark with the appropriate value, for example ?A or ?25 or ?Y. The question mark tells GENCPM to stop and prompt the user for input, then continue automatically. At a ?value entry, GENCPM displays the default value and stops for verification.

The following pages display GENCPM questions. The items in parentheses are the default values. The Question Variable associated with the question is shown below the explanation of the answers to the questions.

Program Questions:

Use GENCPM.DAT for defaults (Y) ?

Enter Y - GENCPM gets its default values from the file GENCPM.DAT.

Enter N - GENCPM uses the built-in default values.

No Question Variable is associated with this question

Create a new GENCPM.DAT file (N) ?

Enter N - GENCPM does not create a new GENCPM.DAT file.

Enter Y - After GENCPM generates the new CPM3.SYS file it creates a new GENCPM.DAT file containing the default values.

Question Variable: CRDATAF

Display Load Table at Cold Boot (Y) ?

Enter Y - On Cold Boot the system displays the load table containing the filename, filetype, hex starting address, length of system modules, and the TPA size.

Enter N - System displays only the TPA size on cold boot.

Question Variable: PRTMSG

Number of console columns (#80) ?

Enter the number of columns (characters-per-line) for your console.

A character in the last column must not force a new line for console editing in CP/M 3. If your terminal forces a new line automatically, decrement the column count by one.

Question Variable: PAGWID

Number of lines per console page (#24) ?

Enter the number of the lines per screen for your console.

Question Variable: PAGLEN

Backspace echoes erased character (N) ?

Enter N - Backspace (Ctrl-H, 08H) moves back one column and erases the previous character.

Enter Y - Backspace moves forward one column and displays the previous character.

Question Variable: BACKSPC

Rubout echoes erased character (Y) ?

Enter Y - Rubout (7FH) moves forward one column and displays the previous character.

Enter N - Rubout moves back one column and erases the previous character.

Question Variable: RUBOUT

Initial default drive (A:) ?

Enter the drive code the prompt is to display at cold boot.

Question Variable: BOOTDRV

Top page of memory (FF) ?

Enter the page address that is to be the top of the operating system. 0FFH is the top of a 64K system.

Question Variable: MEMTOP

Bank-switched memory (Y) ?

Enter Y - GENCPM uses the banked system files.

Enter N - GENCPM uses the nonbanked system files.

Question Variable: BNKSWT

Common memory base page (C0) ?

This question is displayed only if you answered Y to the previous question. Enter the page address of the start of common memory.

Question Variable: COMBAS

Long error messages (Y) ?

This question is displayed only if you answered Y to bank-switched memory.

Enter Y - CP/M 3 error messages contain the BDOS function number and the name of the file on which the operation was attempted.

Enter N - CP/M 3 error messages do not display the function number or file.

Question Variable: LERROR

Double allocation vectors (Y) ?

This question is displayed only if you answered N to bank-switched memory. For more information about double allocation vectors, see the definition of the Disk Parameter Header ALV field in Section 3.

Enter Y - GENCPM creates double-bit allocation vectors for each drive.

Enter N - GENCPM creates single-bit allocation vectors for each drive.

Question Variable: DBLALV

Accept new system definition (Y) ?

Enter Y - GENCPM proceeds to the next set of questions.

Enter N - GENCPM repeats the previous questions and displays your previous input in the default parentheses. You can modify your answers.

No Question Variable is associated with this question.

Number of memory segments (#3) ?

GENCPM displays this question if you answered Y to bank-switched memory.

Enter the number of memory segments in the system. Do not count common memory or memory in Bank 1, the TPA bank, as a memory segment. A maximum of 16 (0 - 15) memory segments are allowed. The memory segments define to GENCPM the memory available for buffer and hash table allocation. Do not include the part of Bank 0 that is reserved for the operating system.

Question Variable: NUMSEGS

CP/M 3 Base,size,bank (8E,32,00)

Enter memory segment table:
 Base,size,bank (00,8E,00) ?
 Base,size,bank (00,C0,02) ?
 Base,size,bank (00,C0,03) ?

Enter the base page, the length, and the bank of the memory segment.

Question Variable: MEMSEG0# where # = 0 to F hex

Accept new memory segment table entries (Y) ?

Enter Y - GENCPM displays the next group of questions.

Enter N - GENCPM displays the memory segment table definition questions again.

No Question Variable is associated with this question.

Setting up directory hash tables:

Enable hashing for drive d: (Y) :

GENCPM displays this question if there is a Drive Table and if the DPHs for a given drive have an 0FFFEH in the hash table address field of the DPH. The question is asked for every drive d: defined in the BIOS.

Enter Y - Space is allocated for the Hash Table. The address and bank of the Hash Table is entered into the DPH.

Enter N - No space is allocated for a Hash Table for that drive.

Question Variable: HASHDRVd where d = drives A-P.

Setting up Blocking/Deblocking buffers:

GENCPM displays the next set of questions if either or both the DTABCB field or the DIRBCB field contain 0FFFEH.

Number of directory buffers for drive d: (#1) ? 10

This question appears only if you are generating a banked system. Enter the number of directory buffers to allocate for the specified drive. In a banked system, directory buffers are allocated only inside Bank 0. In a nonbanked system, one directory buffer is allocated above the BIOS.

Question Variable: NDIRRECd where d = drives A-P.

Number of data buffers for drive d: (#1) ? 1

This question appears only if you are generating a Banked system. Enter the number of data buffers to allocate for the specified drive. In a banked system, data buffers can only be allocated outside Bank 1, and in common. You can only allocate data buffers in alternate banks if your BIOS supports interbank moves. In a nonbanked system, data buffers are allocated above the BIOS.

Question Variable: NDTARECd where d = drives A-P.

Share buffer(s) with which drive (A:) ?

This question appears only if you answered zero to either of the above questions. Enter the drive letter (A-P) of the drive with which you want this drive to share a buffer.

Question Variable: ODIRDRVd for directory records where d = drives A-P.

Question Variable: ODTADRVd for data records where d = drives A-P.

Allocate buffers outside of Commom (N) ?

This question appears if the BIOS XMOVE routine is implemented.

Answer Y - GENCPM allocates data buffers outside of common and Bank 0.

Answer N - GENCPM allocates data buffers in common.

Question Variable: ALTBNKSd where d = drives A-P.

Overlay Directory buffer for drive d: (Y) ?

This question appears only if you are generating a nonbanked system.

Enter Y - this drive shares a directory buffer with another drive.

Enter N - GENCPM allocates an additional directory buffer above the BIOS.

Question Variable: OVLYDIRd where d = drives A-P.

Overlay Data buffer for drive d: (Y) ?

This question appears only if you are generating a nonbanked system.

Enter Y - this drive shares a data buffer with another drive.

Enter N - GENCPM allocates an additional data buffer above the BIOS.

Question Variable: OVLYDTAd for directory records where d = drives A-P.

Accept new buffer definitions (Y) ?

Enter Y - GENCPM creates the CPM3.SYS file and terminates.

Enter N - GENCPM redisplays all of the buffer definition questions.

No Question Variable is associated with this question.

Examples:

The following section contains examples of two system generation sessions. If no entry follows a program question, assume RETURN was entered to select the default value in parentheses. Entries different from the default appear after the question mark.

EXAMPLE OF CONTENTS OF GENCPM.DAT FILE

```
combas = c0 <CR>
lerror = ? <CR>
numsegs = 3 <CR>
memseg00 = 00,80,00 <CR>
memseg01 = 0d,b3,02 <CR>
memseg0f = ?00,c0,10 <CR>
hashdrva = y <CR>
hashdrvd = n <CR>
ndirreca = 20 <CR>
ndtarecf = 10 <CR>
```

EXAMPLE OF SYSTEM GENERATION WITH BANKED MEMORY

A>**GENCPM**

```
CP/M 3.0 System Generation
Copyright (C) 1982, Digital Research

Default entries are shown in (parens).
Default base is Hex, precede entry with # for decimal
```

```
Use GENCPM.DAT for defaults (Y) ?

Create a new GENCPM.DAT file (N) ?

Display Load Map at Cold Boot (Y) ?

Number of console columns (#80) ?
Number of lines in console page (#24) ?
Backspace echoes erased character (N) ?
Rubout echoes erased character (N) ?

Initial default drive (A:) ?

Top page of memory (FF) ?
Bank switched memory (Y) ?
Common memory base page (C0) ?

Long error messages (Y) ?

Accept new system definition (Y) ?

Setting up Allocation vector for drive A:
Setting up Checksum vector for drive A:
Setting up Allocation vector for drive B:
Setting up Checksum vector for drive B:
Setting up Allocation vector for drive C:
Setting up Checksum vector for drive C:
Setting up Allocation vector for drive D:
Setting up Checksum vector for drive D:

*** Bank 1 and Common are not included ***
*** in the memory segment table.       ***

Number of memory segments (#3) ?

CP/M 3 Base,size,bank (8B,35,00)

Enter memory segment table:
 Base,size,bank (00,8B,00) ?
 Base,size,bank (0D,B3,02) ?
 Base,size,bank (00,C0,03) ?

 CP/M 3 Sys    8B00H 3500H  Bank 00
 Memseg No. 00 0000H 8B00H  Bank 00
 Memseg No. 01 0D00H B300H  Bank 02
 Memseg No. 02 0000H C000H  Bank 03

Accept new memory segment table entries (Y) ?

Setting up directory hash tables:
 Enable hashing for drive A: (Y) ?
 Enable hashing for drive B: (Y) ?
 Enable hashing for drive C: (Y) ?
 Enable hashing for drive D: (Y) ?
```

```
Setting up Blocking/Deblocking buffers:

The physical record size is 0200H:

     Available space in 256 byte pages:
     TPA = 00F4H, Bank 0 = 008BH, Other banks = 0166H

               Number of directory buffers for drive A: (#32) ?

     Available space in 256 byte pages:
     TPA = 00F4H, Bank 0 = 0049H, Other banks = 0166H

               Number of data buffers for drive A: (#2) ?
               Allocate buffers outside of Common (N) ?

     Available space in 256 byte pages:
     TPA = 00F0H, Bank 0 = 0049H, Other banks = 0166H

               Number of directory buffers for drive B: (#32) ?

     Available space in 256 byte pages:
     TPA = 00F0H, Bank 0 = 0007H, Other banks = 0166H

               Number of data buffers for drive B: (#0) ?
               Share buffer(s) with which drive (A:) ?

The physical record size is 0080H:

     Available space in 256 byte pages:
     TPA = 00F0H, Bank 0 = 0007H, Other banks = 0166H

               Number of directory buffers for drive C: (#10) ?

     Available space in 256 byte pages:
     TPA = 00F0H, Bank 0 = 0001H, Other banks = 0166H

               Number of directory buffers for drive D: (#0) ?
               Share buffer(s) with which drive (C:) ?

     Available space in 256 byte pages:
     TPA = 00F0H, Bank 0 = 0001H, Other banks = 0166H


Accept new buffer definitions (Y) ?

 BNKBIOS3 SPR  F600H  0600H
 BNKBIOS3 SPR  B100H  0F00H
 RESBDOS3 SPR  F000H  0600H
 BNKBDOS3 SPR  8700H  2A00H

*** CP/M 3.0 SYSTEM GENERATION DONE ***
```

In the preceding example GENCPM displays the resident portion of BNKBIOS3.SPR first, followed by the banked portion.

EXAMPLE OF SYSTEM GENERATION WITH NONBANKED MEMORY

```
A>GENCPM

CP/M 3.0 System Generation
Copyright (C) 1982, Digital Research

Default entries are shown in (parens).
Default base is Hex, precede entry with # for decimal

Use GENCPM.DAT for defaults (Y) ?

Create a new GENCPM.DAT file (N) ?

Display Load Map at Cold Boot (Y) ?

Number of console columns (#80) ?
Number of lines in console page (#24) ?
Backspace echoes erased character (N) ?
Rubout echoes erased character (N) ?

Initial default drive (A:) ?

Top page of memory (FF) ?
Bank switched memory (Y) ? N

Double allocation vectors (Y) ?

Accept new system definition (Y) ?

Setting up Blocking/Deblocking buffers:

The physical record size is 0200H:

     Available space in 256 byte pages:
     TPA = 00D8H

     *** Directory buffer required  ***
     *** and allocated for drive A: ***

     Available space in 256 byte pages:
     TPA = 00D5H

               Overlay Data buffer for drive A: (Y) ?

     Available space in 256 byte pages:
     TPA = 00D5H

               Overlay Directory buffer for drive B: (Y) ?
               Share buffer(s) with which drive (A:) ?

     Available space in 256 byte pages:
     TPA = 00D5H
```

```
                  Overlay Data buffer for drive B: (Y) ?
                  Share buffer(s) with which drive (A:) ?

   The physical record size is 0080H:

        Available space in 256 byte pages:
        TPA = 00D5H

                  Overlay Directory buffer for drive C: (Y) ?
                  Share buffer(s) with which drive (A:) ?

        Available space in 256 byte pages:
        TPA = 00D5H

                  Overlay Directory buffer for drive D: (Y) ?
                  Share buffer(s) with which drive (C:) ?

        Available space in 256 byte pages:
        TPA = 00D5H


   Accept new buffer definitions (Y) ?

    BIOS3     SPR  F300H  0B00H
    BDOS3     SPR  D600H  1D00H

   *** CP/M 3.0 SYSTEM GENERATION DONE ***

   A>
```

## 5.2 Customizing the CPMLDR

The CPMLDR resides on the system tracks of a CP/M 3 system disk, and loads the CPM3.SYS file into memory to cold start the system. CPMLDR contains the LDRBDOS supplied by Digital Research, and must contain your customized LDRBIOS.

The system tracks for CP/M 3 contain the customized Cold Start Loader, CPMLDR with the customized LDRBIOS, and possibly the CCP.

The COPYSYS utility places the Cold Start Loader, the CPMLDR, and optionally the CCP on the system tracks, as shown in Table 5-1.

**Table 5-1. Sample CP/M 3 System Track Organization**

| Track | Sector | Page | Memory Address | CP/M 3 Module Name |
|---|---|---|---|---|
| 00 | 01 | | Boot Address | Cold Start Loader |
| 00 | 02 | 00 | 0100H | CPMLDR |
| . | | | | |
| . | | | | and |
| . | | | | |
| 00 | 21 | 09 | 0A80H | LDRBDOS |
| 00 | 22 | 10 | 0B00H | LDRBIOS |
| . | | | | |
| . | | | | |
| 00 | 26 | 12 | 0D00H | and |
| 01 | 01 | 12 | 0D80H | |
| . | | | | |
| . | | | | |
| 01 | 26 | 25 | 1A00H | CCP |

Typically the Cold Start Loader is loaded into memory from Track 0, Sector 1 of the system tracks when the reset button is depressed. The Cold Start Loader then loads CPMLDR from the system tracks into memory.

Alternatively, if you are starting from an existing CP/M 2 system, you can run CPMLDR.COM as a transient program. CP/M 2 loads CPMLDR.COM into memory at location 100H. CPMLDR then reads the CPM3.SYS file from User 0 on drive A and loads it into memory.

Use the following procedure to create a customized CPMLDR.COM file, including your customized LDRBIOS:

1) Prepare a LDRBIOS.ASM file.

2) Assemble the LDRBIOS file with RMAC to produce a LDRBIOS.REL file.

3) Link the supplied CPMLDR.REL file with the LDRBIOS.REL file you created to produce a CPMLDR.COM file.

   A>**LINK CPMLDR[L100]=CPMLDR,LDRBIOS**

   Replace the address 100 with the load address to which your boot loader loads CPMLDR.COM. You must include a bias of 100H bytes for buffer space when you determine the load address.

The CPMLDR requires a customized LDRBIOS to perform disk input and console output. The LDRBIOS is essentially a nonbanked BIOS. The LDRBIOS has the same JMP vector as the regular CP/M 3 BIOS. The LDRBIOS is called only to perform disk reads (READ) from one drive, console output (CONOUT) for sign-on messages, and minimal system initialization.

The CPMLDR calls the BOOT entry point at the beginning of the LDRBIOS to allow it to perform any necessary hardware initialization. The BOOT entry point should return to CPMLDR instead of loading and branching to the CCP, as a BIOS normally does. Note that interrupts are not disabled when the LDRBIOS BOOT routine is called.

Test your LDRBIOS completely to ensure that it properly performs console character output and disk reads. Check that the proper tracks and sectors are addressed on all reads and that data is transferred to the proper memory locations.

You should assemble the LDRBIOS.ASM file with a relocatable origin of 0000H. Assemble the LDRBIOS with RMAC to produce a LDRBIOS.REL file. Link the LDRBIOS.REL file with the CPMLDR.REL file supplied by Digital Research to create a CPMLDR.COM file. Use the L option in LINK to specify the load origin (address) to which the boot loader on track 0 sector 1 loads the CPMLDR.COM file.

Unnecessary BIOS functions can be deleted from the LDRBIOS to conserve space. There is one absolute restriction on the length of the LDRBIOS: it cannot extend above the base of the banked portion of CP/M 3. (GENCPM lists the base address of CP/M 3 in its load map.) If you plan to boot CP/M 3 from standard, single-density, eight-inch floppy disks, your CPMLDR must not be longer than 1980H to place the CPMLDR.COM file on two system tracks with the boot sector. If the CCP resides on the system tracks with the Cold Start Loader and CPMLDR, the combined lengths must not exceed 1980H.

## 5.3 CPMLDR Utility

Syntax:

    CPMLDR

Purpose:

CPMLDR loads the CP/M 3 system file CPM3.SYS into Bank 0 and transfers control to the BOOT routine in the customized BIOS. You can specify in GENCPM for CPMLDR to display a load table containing the names and addresses of the system modules.

The CPM3.SYS file contains the CP/M 3 BDOS and customized BIOS. The file CPM3.SYS must be on drive A in USER 0. You can execute CPMLDR under SID™ or DDT™ to help debug the BIOS. A $B in the default File Control Block (FCB) causes CPMLDR to execute a RST 7 (SID breakpoint) just before jumping to the CP/M 3 Cold Boot BIOS entry point.

Input File:

```
CPM3.SYS
```

Examples:

```
A>CPMLDR
CP/M V3.0 Loader
Copyright (C) 1982, Digital Research

 BNKBIOS3 SPR  F600H  0A00H
 BNKBIOS3 SPR  BB00H  0500H
 RESBDOS3 SPR  F100H  0500H
 BNKBDOS3 SPR  9A00H  2100H

 60K TPA
A>
```

In the preceding example, CPMLDR displays its name and version number, the Digital Research copyright message, and a four-column load table containing the filename, filetype, hex starting address, and length of the system modules. CPMLDR completes its sign-on message by indicating the size of the Transient Program Area (TPA) in kilobytes. The CCP then displays the system prompt, A>.

## 5.4 Booting CP/M 3

The CP/M 3 cold start operation loads the CCP, BDOS, and BIOS modules into their proper locations in memory and passes control to the cold start entry point (BIOS Function 0: BOOT) in the BIOS. Typically, a PROM-based loader initiates a cold start by loading sector 0 on track 1 of the system tracks into memory and jumping to it. This first sector contains the Cold Start Loader. The Cold Start Loader loads the CPMLDR.COM program into memory and jumps to it. CPMLDR loads the CPM3.SYS file into memory and jumps to the BIOS cold start entry point.

To boot the CP/M 3 system, use the following procedure:

1) Create the CPM3.SYS file.

2) Copy the CPM3.SYS file to the boot drive.

3) Create a CPMLDR.COM for your machine.

4) Place the CPMLDR.COM file on your system tracks using SYSGEN with CP/M 2 or COPYSYS with CP/M 3. The boot loader must place the CPMLDR.COM file at the address at which it originated. If CPMLDR has been linked to load at 100H, you can run CPMLDR under CP/M 2.

The COPYSYS utility handles initialization of the system tracks. The source of COPYSYS is included with the standard CP/M 3 system because you need to customize COPYSYS to support nonstandard system disk formats. COPYSYS copies the Cold Start Loader, the CPMLDR.COM file, and optionally the CCP to the system tracks. Refer to the COPYSYS.ASM source file on the distribution disk.

End of Section 5

# Section 6
# Debugging the BIOS

This section describes a sample debugging session for a nonbanked CP/M 3 BIOS. You must create and debug your nonbanked system first, then bring up the banked system. Note that your system probably displays addresses that differ from the addresses in the following example.

You can use SID, Digital Research's Symbolic Debugger Program, running under CP/M 2.2, to help debug your customized BIOS. The following steps outline a sample debugging session.

1) Determine the amount of memory available to CP/M 3 when the debugger and CP/M 2.2 are in memory. To do this, load the debugger under CP/M 2.2 and list the jump instruction at location 0005H. In the following example of a 64K system, C500 is the base address of the debugger, and also the maximum top of memory that you can specify in GENCPM for your customized CP/M 3 system.

```
A>SID
CP/M 3 SID - Version 3.0
#L5
0005  JMP  C500
        .
        .
        .
```

2) Running under CP/M 2.2, use GENCPM to generate a CPM3.SYS file, which specifies a top of memory that is less than the base address of the debugger, as determined by the previous step. Allow at least 256K bytes for a patch area. In this example, you can specify C3 to GENCPM as the top of memory for your CP/M 3 system.

```
A>GENCPM
.
.
.
Top page of memory (FF)?  C3
.
.
.
```

3) Now you have created a system small enough to debug under SID. Use SID to load the CPMLDR.COM file, as shown in the

following example:

```
A>SID CPMLDR.COM
CP/M 3 SID - Version 3.0
NEXT MSZE  PC  END
0E80 0E80 0100 D4FF
   #
```

4) Use the I command in SID, as shown in the next example, to place the characters $B into locations 005DH and 005EH of the default FCB based at 005CH. The $B causes CPMLDR.COM to break after loading the CPM3.SYS file into memory.

```
#I$B
```

5) Transfer control to CPMLDR using the G command:

```
#G
```

At this point, the screen clears and the following information appears:

```
CP/M  V3.0 LOADER
Copyright (c) 1982, Digital Research

 BIOS3    SPR  AA00  0B00
 BDOS3    SPR  8B00  1F00

 34K  TPA

* 01A9
#
```

6) With the CP/M 3 system in the proper location, you can set passpoints in your BIOS. Use the L command with the address specified as the beginning of the BIOS by the CPMLDR load table as shown in step 5 above. This L command causes SID to display the BIOS jump vector which begins at that address. The jump vector indicates the beginning address of each subroutine in the table. For example, the first jump instruction in the example below is to the Cold Boot subroutine.

```
#LAA00
```

The output from your BIOS might look like this:

```
JMP AA68
JMP AA8E
JMP ABA4
JMP ABAF
JMP ABCA
     .
     .
     .
```

7) Now set a passpoint in the Cold BOOT routine. Use the P command with an address to set a passpoint at that address.

   **#PAA68**

8) Continue with the CPMLDR.COM program by entering the G command, followed by the address of Cold Boot, the first entry in the BIOS jump vector.

   **#GAA00**

9) In response to the G command, the CPMLDR transfers control to the CP/M 3 operating system. If you set a passpoint in the Cold BOOT routine, the program stops executing, control transfers to SID, and you can begin tracing the BOOT routine.

10) When you know the BOOT routine is functioning correctly, enter passpoints for the other routines you want to trace, and begin tracing step by step to determine the location of problems.

Refer to the Digital Research Symbolic Instruction Debugger User's Guide (SID) in the Programmer's Utilities Guide for the CP/M Family of Operating Systems for a discussion of all the SID commands.

End of Section 6

# Appendix A
# Removable Media Considerations

All disk drives under CP/M 3 are classified as either permanent or removable. In general, removable drives support media changes; permanent drives do not. Setting the high-order bit in the CKS field in a drive's Disk Parameter Block (DPB) marks the drive as a permanent drive.

The BDOS file system distinguishes between permanent and removable drives. If a drive is permanent, the BDOS always accepts the contents of physical record buffers as valid. In addition, it also accepts the results of hash table searches on the drive.

On removable drives, the status of physical record buffers is more complicated. Because of the potential for media change, the BDOS must discard directory buffers before performing most directory related BDOS function calls. This is required because the BDOS detects media changes by reading directory records. When it reads a directory record, the BDOS computes a checksum for the record, and compares the checksum to the currently stored value in the drive's checksum vector. If the checksum values do not match, the BDOS assumes the media has changed. Thus, the BDOS can only detect a media change by an actual directory READ operation.

A similar situation occurs with directory hashing on removable drives. Because the directory hash table is a memory-resident table, the BDOS must verify all unsuccessful hash table searches on removable drives by accessing the directory.

The net result of these actions is that there is a significant performance penalty associated with removable drives as compared to permanent drives. In addition, the protection provided by classifying a drive as removable is not total. Media changes are only detected during directory operations. If the media is changed on a drive during BDOS WRITE operations, the new disk can be damaged.

The BIOS media flag facility gives you another option for supporting drives with removable media. However, to use this option, the disk controller must be capable of generating an interrupt when the drive door is opened. If your hardware provides this support, you can improve the handling of removable media by implementing the following procedure:

1) Mark the drive as a permanent drive and set the DPB CKS parameter to the total number of directory entries, divided by four. For example, set the CKS field for a disk with 96 directory entries to 8018H.

2) Implement an interrupt service routine that sets the @MEDIA flag in the System Control Block and the DPH MEDIA byte for the drive that signaled the door open condition.

By using the media flag facility, you gain the performance advantage associated with permanent drives on drives that support removable media. The BDOS checks the System Control Block @MEDIA flag on entry for all disk-related function calls. If the flag has not been set, it implies that no disks on the system have been changed. If the flag is set, the BDOS checks the DPH MEDIA flag of each currently logged-in disk. If the DPH MEDIA flag of a drive is set, the BDOS reads the entire directory on the drive to determine whether the drive has had a media change before performing any other operations on the drive. In addition, it temporarily classifies any permanent disk with the DPH MEDIA flag set as a removable drive. Thus, the BDOS discards all directory physical record buffers when a drive door is opened to force all directory READ operations to access the disk.

To summarize, using the BIOS MEDIA flag with removable drives offers two important benefits. First, because a removable drive can be classified as permanent, performance is enhanced. Second, because the BDOS immediately checks the entire directory before performing any disk-related function on the drive if the drive's DPH MEDIA flag is set, disk integrity is enhanced.

End of Appendix A

# Appendix B
# Auto-density Support

Auto-density support refers to the capability of CP/M 3 to support different types of media on a single drive. For example, some floppy-disk drives accept single-sided and double-sided disks in both single-density and double-density formats. Auto-density support requires that the BIOS be able to determine the current density when SELDSK is called and to subsequently be able to detect a change in disk format when the READ or WRITE routines are called.

To support multiple disk formats, the drive's BIOS driver must include a Disk Parameter Block (DPB) for each type of disk or include code to generate the proper DPB parameters dynamically. In addition, the BIOS driver must determine the proper format of the disk when the SELDSK entry point is called with register E bit 0 equal to 0 (initial SELDSK calls). If the BIOS driver cannot determine the format, it can return 0000H in register pair HL to indicate the select was not successful. Otherwise, it must update the Disk Parameter Header (DPH) to address a DPB that describes the current media, and return the address of the DPH to the BDOS.

**Note:** all subsequent SELDSK calls with register E bit 0 equal to 1, the BIOS driver must continue to return the address of the DPH returned in the initial SELDSK call. The value 0000H is only a legal return value for initial SELDSK calls.

After a driver's SELDSK routine has determined the format of a disk, the driver's READ and WRITE routines assume this is the correct format until an error is detected. If an error is detected and the driver determines that the media has been changed to another format, it must return the value 0FFH in register A and set the media flag in the System Control Block. This signals the BDOS that the media has changed and the next BIOS call to the drive will be an initial SELDSK call. Do not modify the drive's DPH or DPB until the initial SELDSK call is made. Note that the BDOS can detect a change in media and will make an initial SELDSK call, even though the BIOS READ and WRITE routines have not detected a disk format change. However, the SELDSK routine must always determine the format on initial calls.

A drive's Disk Parameter Header (DPH) has associated with it several uninitialized data areas: the allocation vector, the checksum vector, the directory hash table, and physical record buffers. The size of these areas is determined by DPB parameters. If space for these areas is explicitly allocated in the BIOS, the DPB that requires the most space determines the amount of memory to allocate. If the BIOS defers the allocation of these areas to GENCPM, the DPH must be initialized to the DPB with the largest space requirements. If one DPB is not largest in all of the above categories, a false one must be constructed so that GENCPM allocates sufficient space for each data area.

End of Appendix B

# Appendix C
# Modifying a CP/M 2 BIOS

If you are modifying an existing CP/M 2.2 BIOS, you must note the following changes.

- The BIOS jump vector is expanded from 17 entry points in CP/M 2.2 to 33 entry points in CP/M 3. You must implement the necessary additional routines.
- The Disk Parameter Header and Disk Parameter Block data structures are expanded.

See Section 3 of this manual, "CP/M 3 BIOS Functional Specifications," for details of the BIOS data structures and subroutines. The following table shows all CP/M 3 BIOS functions with the changes necessary to support CP/M 3.

**Table C-1. CP/M 3 BIOS Functions**

| Function | Meaning |
|---|---|
| BIOS Function 00: | BOOT |
| | The address for the JMP at location 5 must be obtained from @MXTPA in the System Control Block. |
| BIOS Function 01: | WBOOT |
| | The address for the JMP at location 5 must be obtained from @MXTPA in the System Control Block. The CCP can be reloaded from a file. |
| BIOS Function 02: | CONST |
| | Can be implemented unchanged. |
| BIOS Function 03: | CONIN |
| | Can be implemented unchanged. Do not mask the high-order bit. |

**Table C-1. (continued)**

| Function | Meaning |
|---|---|
| BIOS Function 04: | CONOUT<br>Can be implemented unchanged. |
| BIOS Function 05: | LIST<br>Can be implemented unchanged. |
| BIOS Function 06: | AUXOUT<br>Called PUNCH in CP/M 2. Can be implemented unchanged. |
| BIOS Function 07: | AUXIN<br>Called READER in CP/M 2. Can be implemented unchanged. Do not mask the high-order bit. |
| BIOS Function 08: | HOME<br>No change. |
| BIOS Function 09: | SELDSK<br>Can not return a select error when SELDSK is called with bit 0 in register E equal to 1. |
| BIOS Function 10: | SETTRK<br>No change. |
| BIOS Function 11: | SETSEC<br>Sectors are physical sectors, not logical 128-byte sectors. |
| BIOS Function 12: | SETDMA<br>Now called for every READ or WRITE operation. The DMA buffer can now be greater than 128 bytes. |

**Table C-1. (continued)**

| Function | Meaning |
|---|---|
| BIOS Function 13: | READ |
| | READ operations are in terms of physical sectors. READ can return a 0FFH error code if it detects that the disk format has changed. |
| BIOS Function 14: | WRITE |
| | WRITE operations are in terms of physical sectors. If write detects that the disk is Read-Only, it can return error code 2. WRITE can return a 0FFH error code if it detects that the disk format has changed. |
| BIOS Function 15: | LISTST |
| | Can be implemented unchanged. |
| BIOS Function 16: | SECTRN |
| | Sectors are physical sectors, not logical 128-byte sectors. |

The following is a list of new BIOS functions:

BIOS Function 17: CONOST

BIOS Function 18: AUXIST

BIOS Function 19: AUXOST

BIOS Function 20: DEVTBL

BIOS Function 21: DEVINI

BIOS Function 22: DRVTBL

BIOS Function 23: MULTIO

BIOS Function 24: FLUSH

BIOS Function 25: MOVE

BIOS Function 26: TIME

BIOS Function 27: SELMEM

BIOS Function 28: SETBNK

BIOS Function 29: XMOVE

BIOS Function 30: USERF

BIOS Function 31: RESERV1

BIOS Function 32: RESERV2

End of Appendix C

# Appendix D
# CPM3.SYS File Format

**Table D-1. CPM3.SYS File Format**

| Record | Contents |
|---|---|
| 0 | Header Record (128 bytes) |
| 1 | Print Record (128 bytes) |
| 2-n | CP/M 3 operating system in reverse order, top down. |

**Table D-2. Header Record Definition**

| Byte | Contents |
|---|---|
| 0 | Top page plus one, at which the resident portion of CP/M 3 is to be loaded top down. |
| 1 | Length in pages (256 bytes) of the resident portion of CP/M 3. |
| 2 | Top page plus one, at which the banked portion of CP/M 3 is to be loaded top down. |
| 3 | Length in pages (256 bytes) of the banked portion of CP/M 3. |
| 4-5 | Address of CP/M 3 Cold Boot entry point. |
| 6-15 | Reserved. |
| 16-51 | Copyright Message. |
| 52 | Reserved. |
| 53-58 | Serial Number. |
| 59-127 | Reserved. |

The Print Record is the CP/M 3 Load Table in ASCII, terminated by a dollar sign ($).

End of Appendix D

# Appendix E
# Root Module of Relocatable BIOS for CP/M 3

All the listings in Appendixes E through I are assembled with RMAC, the CP/M Relocating Macro Assembler, and cross-referenced with XREF™ , an assembly language cross-reference program used with RMAC. These listings are output from the XREF program. The assembly language sources are on your distribution disk as .ASM files.

```
                            title   'Root module of relocatable BIOS for CP/M 3.0'

                            ; version 1.0 15 Sept 82

 FFFF =          true       equ -1
 0000 =          false      equ not true

 FFFF =          banked     equ true


                 ;                  Copyright (C), 1982
                 ;                 Digital Research, Inc
                 ;                     P.O. Box 579
                 ;                Pacific Grove, CA  93950


                 ;  This is the invariant portion of the modular BIOS and is
                 ;      distributed as source for informational purposes only.
                 ;      All desired modifications should be performed by
                 ;      adding or changing externally defined modules.
                 ;      This allows producing "standard" I/O modules that
                 ;      can be combined to support a particular system
                 ;      configuration.

 000D =          cr         equ 13
 000A =          lf         equ 10
 0007 =          bell       equ 7
 0011 =          ctlQ       equ 'Q'-'@'
 0013 =          ctlS       equ 'S'-'@'

 0100 =          ccp        equ 0100h       ; Console Command Processor gets loaded into the TPA

                            cseg            ; GENCPM puts CSEG stuff in common memory


                    ; variables in system data page

                            extrn @covec,@civec,@aovec,@aivec,@lovec ; I/O redirection vectors
                            extrn @mxtpa                            ; addr of system entry point
                            extrn @bnkbf                            ; 128 byte scratch buffer

                    ; initialization

                            extrn ?init                 ; general initialization and signon
                            extrn ?ldccp,?rlccp         ; load & reload CCP for BOOT & WBOOT

                    ; user defined character I/O routines

                            extrn ?ci,?co,?cist,?cost   ; each take device in <B>
                            extrn ?cinit                ; (re)initialize device in <C>
                            extrn @ctbl                 ; physical character device table

                    ; disk communication data items

                            extrn @dtbl                 ; table of pointers to XDPHs
                            public @adrv,@rdrv,@trk,@sect   ; parameters for disk I/O
                            public @dma,@dbnk,@cnt      ;    ''         ''   ''  ''

                    ; memory control
```

**Listing E-1. Root Module of Relocatable BIOS for CP/M 3**

```
                                public @cbnk                    ; current bank
                                extrn ?xmove,?move              ; select move bank, and block move
                                extrn ?bank                     ; select CPU bank

                            ; clock support

                                extrn ?time                     ; signal time operation

                            ; general utility routines

                                public ?pmsg,?pdec      ; print message, print number from 0 to 65535
                                public ?pderr           ; print BIOS disk error message header

                                maclib modebaud         ; define mode bits


                            ; External names for BIOS entry points

                                public ?boot,?wboot,?const,?conin,?cono,?list,?auxo,?auxi
                                public ?home,?sldsk,?sttrk,?stsec,?stdma,?read,?write
                                public ?lists,?sctrn
                                public ?conos,?auxis,?auxos,?dvtbl,?devin,?drtbl
                                public ?mltio,?flush,?mov,?tim,?bnksl,?stbnk,?xmov


                            ; BIOS Jump vector.

                                        ; All BIOS routines are invoked by calling these
                                        ;       entry points.

 0000 C30000        ?boot:  jmp boot        ; initial entry on cold start
 0003 C36C00        ?wboot: jmp wboot       ; reentry on program exit, warm start

 0006 C37701        ?const: jmp const       ; return console input status
 0009 C39201        ?conin: jmp conin       ; return console input character
 000C C3DA00        ?cono:  jmp conout      ; send console output character
 000F C3E600        ?list:  jmp list        ; send list output character
 0012 C3E000        ?auxo:  jmp auxout      ; send auxilliary output character
 0015 C39801        ?auxi:  jmp auxin       ; return auxilliary input character

 0018 C36E00        ?home:  jmp home        ; set disks to logical home
 001B C33F00        ?sldsk: jmp seldsk      ; select disk drive, return disk parameter info
 001E C37100        ?sttrk: jmp settrk      ; set disk track
 0021 C37700        ?stsec: jmp setsec      ; set disk sector
 0024 C37D00        ?stdma: jmp setdma      ; set disk I/O memory address
 0027 C39400        ?read:  jmp read        ; read physical block(s)
 002A C3AA00        ?write: jmp write       ; write physical block(s)

 002D C31201        ?lists: jmp listst      ; return list device status
 0030 C38900        ?sctrn: jmp sectrn      ; translate logical to physical sector

 0033 C30601        ?conos: jmp conost      ; return console output status
 0036 C37D01        ?auxis: jmp auxist      ; return aux input status
 0039 C30C01        ?auxos: jmp auxost      ; return aux output status
 003C C3D200        ?dvtbl: jmp devtbl      ; return address of device def table
 003F C30000        ?devin: jmp ?cinit      ; change baud rate of device

 0042 C3D600        ?drtbl: jmp getdrv      ; return address of disk drive table
 0045 C3CB00        ?mltio: jmp multio      ; set multiple record count for disk I/O
 0048 C3CF00        ?flush: jmp flush       ; flush BIOS maintained disk caching

 004B C30000        ?mov:   jmp ?move       ; block move memory to memory
 004E C30000        ?tim:   jmp ?time       ; Signal Time and Date operation
 0051 C32502        ?bnksl: jmp bnksel      ; select bank for code execution and default DMA
 0054 C38500        ?stbnk: jmp setbnk      ; select different bank for disk I/O DMA operations.
 0057 C30000        ?xmov:  jmp ?xmove      ; set source and destination banks for one operation

 005A C30000                jmp 0           ; reserved for system implementor
 005D C30000                jmp 0           ; reserved for future expansion
 0060 C30000                jmp 0           ; reserved for future expansion


                            ; BOOT
                            ;       Initial entry point for system startup.

                            dseg    ; this part can be banked

                    boot:
 0000 31D200                lxi sp,boot$stack
 0003 0E0F                  mvi c,15        ; initialize all 16 character devices
                    c$init$loop:
 0005 C5CD0000C1            push b ! call ?cinit ! pop b
 000A 0DF20500              dcr c ! jp c$init$loop
```

**Listing E-1. (continued)**

```
0000E CD0000              call ?init       ; perform any additional system initialization
                                           ; and print signon message

0011 0100102100           lxi b,16*256+0 ! lxi h,@dtbl     ; init all 16 logical disk drives
                  d$init$loop:
0017 C5                   push b           ; save remaining count and abs drive
0018 5E235623             mov e,m ! inx h ! mov d,m ! inx h        ; grab @drv entry
001C 7BB2CA3600           mov a,e ! ora d ! jz d$init$next         ; if null, no drive
0021 E5                   push h                                   ; save @drv pointer
0022 EB                   xchg                                     ; XDPH address in <HL>
0023 2B2B7E32EE           dcx h ! dcx h ! mov a,m ! sta @RDRV      ; get relative drive code
0029 7932ED00             mov a,c ! sta @ADRV                      ; get absolute drive code
002D 2B                   dcx h                                    ; point to init pointer
002E 562B5E               mov d,m ! dcx h ! mov e,m                ; get init pointer
0031 EBCDB601             xchg ! call ipchl                        ; call init routine
0035 E1                   pop h                                    ; recover @drv pointer
                  d$init$next:
0036 C1                   pop b                                    ; recover counter and drive #
0037 0C05C21700           inr c ! dcr b ! jnz d$init$loop          ; and loop for each drive
003C C36300               jmp boot$1

                          cseg   ; following in resident memory

                  boot$1:
0063 CD7800               call set$jumps
0066 CD0000               call ?ldccp                              ; fetch CCP for first time
0069 C30001               jmp ccp


                          ; WBOOT
                          ;       Entry for system restarts.

                  wboot:
006C 31D200               lxi sp,boot$stack
006F CD7800               call set$jumps           ; initialize page zero
0072 CD0000               call ?rlccp              ; reload CCP
0075 C30001               jmp ccp                  ; then reset jmp vectors and exit to ccp


                  set$jumps:

                    if banked
0078 3E01CD5100           mvi a,1 ! call ?bnksl
                    endif

007D 3EC3                 mvi a,JMP
007F 3200003205           sta 0 ! sta 5            ; set up jumps in page zero
0085 2103002201           lxi h,?wboot ! shld 1    ; BIOS warm start entry
008B 2A00002206           lhld @MXTPA ! shld 6     ; BDOS system call entry
0091 C9                   ret


0092                              ds 64
00D2 =            boot$stack      equ $



                          ; DEVTBL
                          ;       Return address of character device table

                  devtbl:
00D2 210000C9             lxi h,@ctbl ! ret


                          ; GETDRV
                          ;       Return address of drive table

                  getdrv:
00D6 210000C9             lxi h,@dtbl ! ret



                          ; CONOUT
                          ;       Console Output.  Send character in <C>
                          ;                        to all selected devices

                  conout:

00DA 2A0000               lhld @covec      ; fetch console output bit vector
00DD C3E900               jmp out$scan
```

**Listing E-1. (continued)**

```
                                ; AUXOUT
                                ;       Auxiliary Output. Send character in <C>
                                ;                    to all selected devices

                        auxout:
00E0 2A0000                     lhld @aovec     ; fetch aux output bit vector
00E3 C3E900                     jmp out$scan


                                ; LIST
                                ;       List Output.  Send character in <C>
                                ;                    to all selected devices.

                        list:
00E6 2A0000                     lhld @lovec     ; fetch list output bit vector

                        out$scan:
00E9 0600                       mvi b,0         ; start with device 0
                        co$next:
00EB 29                         dad h           ; shift out next bit
00EC D2FF00                     jnc not$out$device
00EF E5                         push h          ; save the vector
00F0 C5                         push b          ; save the count and character
                        not$out$ready:
00F1 CD2C01B7CA                 call coster ! ora a ! jz not$out$ready
00F8 C1C5                       pop b ! push b  ; restore and resave the character and device
00FA CD0000                     call ?co        ; if device selected, print it
00FD C1                         pop b           ; recover count and character
00FE E1                         pop h           ; recover the rest of the vector
                        not$out$device:
00FF 04                         inr b           ; next device number
0100 7CB5                       mov a,h ! ora l ; see if any devices left
0102 C2EB00                     jnz co$next     ; and go find them...
0105 C9                         ret


                                ; CONOST
                                ;       Console Output Status.  Return true if
                                ;               all selected console output devices
                                ;               are ready.

                        conost:
0106 2A0000                     lhld @covec     ; get console output bit vector
0109 C31501                     jmp ost$scan


                                ; AUXOST
                                ;       Auxiliary Output Status.  Return true if
                                ;               all selected auxiliary output devices
                                ;               are ready.

                        auxost:
010C 2A0000                     lhld @aovec     ; get aux output bit vector
010F C31501                     jmp ost$scan


                                ; LISTST
                                ;       List Output Status.  Return true if
                                ;               all selected list output devices
                                ;               are ready.

                        listst:
0112 2A0000                     lhld @lovec     ; get list output bit vector

                        ost$scan:
0115 0600                       mvi b,0         ; start with device 0
                        cos$next:
0117 29                         dad h           ; check next bit
0118 E5                         push h          ; save the vector
0119 C5                         push b          ; save the count
011A 3EFF                       mvi a,0FFh      ; assume device ready
011C DC2C01                     cc coster       ; check status for this device
011F C1                         pop b           ; recover count
0120 E1                         pop h           ; recover bit vector
0121 B7                         ora a           ; see if device ready
0122 C8                         rz              ; if any not ready, return false
0123 05                         dcr b           ; drop device number
0124 7CB5                       mov a,h ! ora l ; see if any more selected devices
0126 C21701                     jnz cos$next
0129 F6FF                       ori 0FFh        ; all selected were ready, return true
012B C9                         ret
```

**Listing E-1. (continued)**

```
                    coster:         ; check for output device ready, including optional
                                    ;       xon/xoff support
 012C 682600              mov l,b ! mvi h,0       ; make device code 16 bits
 012F E5                  push h                  ; save it in stack
 0130 292929              dad h ! dad h ! dad h   ; create offset into device characteristics tbl
 0133 11060019            lxi d,@ctbl+6 ! dad d   ; make address of mode byte
 0137 7EE610              mov a,m ! ani mb$xonxoff
 013A E1                  pop h                   ; recover console number in <HL>
 013B CA0000              jz ?cost                ; not a xon device, go get output status direct
 013E 112B0219            lxi d,xofflist ! dad d  ; make pointer to proper xon/xoff flag
 0142 CD5D01              call cistl              ; see if this keyboard has character
 0145 7EC46F01            mov a,m ! cnz cil       ; get flag or read key if any
 0149 FE11C25001          cpi ctlq ! jnz not$q    ; if its a ctl-Q,
 014E 3EFF                mvi a,0FFh              ;       set the flag ready
                    not$q:
 0150 FE13C25701          cpi ctls ! jnz not$s    ; if its a ctl-S,
 0155 3E00                mvi a,00h               ;       clear the flag
                    not$s:
 0157 77                  mov m,a                 ; save the flag
 0158 CD6601              call costl              ; get the actual output status,
 015B A6                  ana m                   ; and mask with ctl-Q/ctl-S flag
 015C C9                  ret                     ; return this as the status

                    cistl:                  ; get input status with <BC> and <HL> saved
 015D C5E5                push b ! push h
 015F CD0000              call ?cist
 0162 E1C1                pop h ! pop b
 0164 B7                  ora a
 0165 C9                  ret

                    costl:                  ; get output status, saving <BC> & <HL>
 0166 C5E5                push b ! push h
 0168 CD0000              call ?cost
 016B E1C1                pop h ! pop b
 016D B7                  ora a
 016E C9                  ret

                    cil:                    ; get input, saving <BC> & <HL>
 016F C5E5                push b ! push h
 0171 CD0000              call ?ci
 0174 E1C1                pop h ! pop b
 0176 C9                  ret


                          ; CONST
                          ;       Console Input Status.  Return true if
                          ;               any selected console input device
                          ;               has an available character.

                    const:
 0177 2A0000              lhld @civec     ; get console input bit vector
 017A C38001              jmp ist$scan


                          ; AUXIST
                          ;       Auxiliary Input Status.  Return true if
                          ;               any selected auxiliary input device
                          ;               has an available character.

                    auxist:
 017D 2A0000              lhld @aivec     ; get aux input bit vector

                    ist$scan:
 0180 0600                mvi b,0         ; start with device 0
                    cis$next:
 0182 29                  dad h           ; check next bit
 0183 3E00                mvi a,0         ; assume device not ready
 0185 DC5D01              cc cistl        ; check status for this device
 0188 B7C0                ora a ! rnz     ; if any ready, return true
 018A 04                  inr b           ; next device number
 018B 7CB5                mov a,h ! ora l ; see if any more selected devices
 018D C28201              jnz cis$next
 0190 AF                  xra a           ; all selected were not ready, return false
 0191 C9                  ret


                          ; CONIN
                          ;       Console Input.  Return character from first
                          ;               ready console input device.

                    conin:
 0192 2A0000              lhld @civec
 0195 C39B01              jmp in$scan
```

**Listing E-1. (continued)**

```


                               ; AUXIN
                               ;       Auxiliary Input.  Return character from first
                               ;               ready auxiliary input device.

                       auxin:
     0198 2A0000               lhld @aivec

                       in$scan:
     019B E5                   push h            ; save bit vector
     019C 0600                 mvi b,0
                       ci$next:
     019E 29                   dad h             ; shift out next bit
     019F 3E00                 mvi a,0           ; insure zero a  (nonexistant device not ready).
     01A1 DC5D01               cc cistl          ; see if the device has a character
     01A4 B7                   ora a
     01A5 C2B201               jnz ci$rdy        ; this device has a character
     01A8 05                   dcr b             ; else, next device
     01A9 7CB5                 mov a,h ! ora l ; see if any more devices
     01AB C29E01               jnz ci$next       ; go look at them
     01AE E1                   pop h             ; recover bit vector
     01AF C39B01               jmp in$scan       ; loop til we find a character

                       ci$rdy:
     01B2 E1                   pop h             ; discard extra stack
     01B3 C30000               jmp ?ci


                       ;       Utility Subroutines


                       ipchl:          ; vectored CALL point
     01B6 E9                   pchl


                       ?pmsg:          ; print message @<HL> up to a null
                                       ; saves <BC> & <DE>
     01B7 C5                   push b
     01B8 D5                   push d
                       pmsg$loop:
     01B9 7EB7CAC801           mov a,m ! ora a ! jz pmsg$exit
     01BE 4FE5                 mov c,a ! push h
     01C0 CD0C00E1             call ?cono ! pop h
     01C4 23C3B901             inx h ! jmp pmsg$loop
                       pmsg$exit:
     01C8 D1                   pop d
     01C9 C1                   pop b
     01CA C9                   ret

                       ?pdec:          ; print binary number 0-65535 from <HL>
     01CB 01F30111F0           lxi b,table10! lxi d,-10000
                       next:
     01D1 3E2F                 mvi a,'0'-1
                       pdecl:
     01D3 E53C19D2DE           push h! inr a! dad d! jnc stoploop
     01D9 3333C3D301           inx sp! inx sp! jmp pdecl
                       stoploop:
     01DE D5C5                 push d! push b
     01E0 4FCD0C00             mov c,a! call ?cono
     01E4 C1D1                 pop b! pop d
                       nextdigit:
     01E6 E1                   pop h
     01E7 0A5F03               ldax b! mov e,a! inx b
     01EA 0A5703               ldax b! mov d,a! inx b
     01ED 7BB2C2D101           mov a,e! ora d! jnz next
     01F2 C9                   ret

                       table10:
     01F3 18FC9CFFF6           dw      -1000,-100,-10,-1,0

                       ?pderr:
     01FD 21D100CDB7           lxi h,drive$msg ! call ?pmsg                    ; error header
     0203 3AED00C641           lda @adrv ! adi 'A' ! mov c,a ! call ?cono      ; drive code
     020C 21E300CDB7           lxi h,track$msg ! call ?pmsg                    ; track header
     0212 2AEF00CDCB           lhld @trk ! call ?pdec                          ; track number
     0218 21E800CDB7           lxi h,sector$msg ! call ?pmsg                   ; sector header
     021E 2AF100CDCB           lhld @sect ! call ?pdec                         ; sector number
     0224 C9                   ret


                               ; BNKSEL
                               ;       Bank Select.  Select CPU bank for further execution.

```

**Listing E-1. (continued)**

```
                        bnksel:
0225 323B02                     sta @cbnk                       ; remember current bank
0228 C30000                     jmp ?bank                       ; and go exit through users
                                                                ; physical bank select routine


022B FFFFFFFFFFxofflist         db      -1,-1,-1,-1,-1,-1,-1,-1         ; ctl-s clears to zero
0233 FFFFFFFFFF                 db      -1,-1,-1,-1,-1,-1,-1,-1



                                dseg    ; following resides in banked memory



                        ;       Disk I/O interface routines


                                ; SELDSK
                                ;       Select Disk Drive.  Drive code in <C>.
                                ;               Invoke login procedure for drive
                                ;               if this is first select.  Return
                                ;               address of disk parameter header
                                ;               in <HL>

                        seldsk:
003F 7932ED00                   mov a,c ! sta @adrv                     ; save drive select code
0043 69260029                   mov l,c ! mvi h,0 ! dad h               ; create index from drive code
0047 01000009                   lxi b,@dtbl ! dad b                     ; get pointer to dispatch table
004B 7E23666F                   mov a,m ! inx h ! mov h,m ! mov l,a     ; point at disk descriptor
004F B4C8                       ora h ! rz                              ; if no entry in table, no disk
0051 7BE601C26D                 mov a,e ! ani 1 ! jnz not$first$select  ; examine login bit
0057 E5EB                       push h ! xchg                           ; put pointer in stack & <DE>
0059 21FEFF197E                 lxi h,-2 ! dad d ! mov a,m ! sta @RDRV  ; get relative drive
0061 21FAFF19                   lxi h,-6 ! dad d                        ; find LOGIN addr
0065 7E23666F                   mov a,m ! inx h ! mov h,m ! mov l,a     ; get address of LOGIN routine
0069 CDB601                     call ipchl                              ; call LOGIN
006C E1                         pop h                                   ; recover DPH pointer
                        not$first$select:
006D C9                         ret


                                ; HOME
                                ;       Home selected drive.  Treated as SETTRK(0).

                        home:
006E 010000                     lxi b,0         ; same as set track zero


                                ; SETTRK
                                ;       Set Track. Saves track address from <BC>
                                ;               in @TRK for further operations.

                        settrk:
0071 6960                       mov l,c ! mov h,b
0073 22EF00                     shld @trk
0076 C9                         ret


                                ; SETSEC
                                ;       Set Sector.  Saves sector number from <BC>
                                ;               in @sect for further operations.

                        setsec:
0077 6960                       mov l,c ! mov h,b
0079 22F100                     shld @sect
007C C9                         ret


                                ; SETDMA
                                ;       Set Disk Memory Address.  Saves DMA address
                                ;               from <BC> in @DMA and sets @DBNK to @CBNK
                                ;               so that further disk operations take place
                                ;               in current bank.

                        setdma:
007D 6960                       mov l,c ! mov h,b
007F 22F300                     shld @dma

0082 3A3B02                     lda @cbnk       ; default DMA bank is current bank
                                                ; fall through to set DMA bank
```

**Listing E-1. (continued)**

```

                            ; SETBNK
                            ;       Set Disk Memory Bank.  Saves bank number
                            ;               in @DBNK for future disk data
                            ;               transfers.

                    setbnk:
0085 32F600                 sta @dbnk
0088 C9                     ret


                            ; SECTRN
                            ;       Sector Translate.  Indexes skew table in <DE>
                            ;               with sector in <BC>.  Returns physical sector
                            ;               in <HL>.  If no skew table (<DE>=0) then
                            ;               returns physical=logical.

                    sectrn:
0089 6960                   mov l,c ! mov h,b
008B 7AB3C8                 mov a,d ! ora e ! rz
008E EB096E2600             xchg ! dad b ! mov l,m ! mvi h,0
0093 C9                     ret


                            ; READ
                            ;       Read physical record from currently selected drive.
                            ;               Finds address of proper read routine from
                            ;               extended disk parameter header (XDPH).

                    read:
0094 2AED002600             lhld @adrv ! mvi h,0 ! dad h    ; get drive code and double it
009A 11000019               lxi d,@dtbl ! dad d             ; make address of table entry
009E 7E23666F               mov a,m ! inx h ! mov h,m ! mov l,a     ; fetch table entry
00A2 E5                     push h                          ; save address of table
00A3 11F8FF19               lxi d,-8 ! dad d                ; point to read routine address
00A7 C3BD00                 jmp rw$common                   ; use common code


                            ; WRITE
                            ;       Write physical sector from currently selected drive.
                            ;               Finds address of proper write routine from
                            ;               extended disk parameter header (XDPH).

                    write:
00AA 2AED002600             lhld @adrv ! mvi h,0 ! dad h    ; get drive code and double it
00B0 11000019               lxi d,@dtbl ! dad d             ; make address of table entry
00B4 7E23666F               mov a,m ! inx h ! mov h,m ! mov l,a     ; fetch table entry
00B8 E5                     push h                          ; save address of table
00B9 11F6FF19               lxi d,-10 ! dad d               ; point to write routine address

                    rw$common:
00BD 7E23666F               mov a,m ! inx h ! mov h,m ! mov l,a     ; get address of routine
00C1 D1                     pop d                           ; recover address of table
00C2 1B1B                   dcx d ! dcx d                   ; point to relative drive
00C4 1A32EE00               ldax d ! sta @rdrv              ; get relative drive code and post it
00C8 1313                   inx d ! inx d                   ; point to DPH again
00CA E9                     pchl                            ; leap to driver


                            ; MULTIO
                            ;       Set multiple sector count. Saves passed count in
                            ;               @CNT

                    multio:
00CB 32F500C9               sta @cnt ! ret


                            ; FLUSH
                            ;       BIOS deblocking buffer flush.  Not implemented.

                    flush:
00CF AFC9                   xra a ! ret             ; return with no error



                            ; error message components
00D1 0D0A074249drive$msg        db      cr,lf,bell,'BIOS Error on ',0
00E3 3A20542D00track$msg        db      ': T-',0
00E8 2C20532D00sector$msg       db      ', S-',0


                        ; disk communication data items
```

**Listing E-1. (continued)**

```
        00ED        @adrv   ds      1               ; currently selected disk drive
        00EE        @rdrv   ds      1               ; controller relative disk drive
        00EF        @trk    ds      2               ; current track number
        00F1        @sect   ds      2               ; current sector number
        00F3        @dma    ds      2               ; current DMA address
        00F5 00     @cnt    db      0               ; record count for multisector transfer
        00F6 00     @dbnk   db      0               ; bank for DMA operations


                            cseg    ; common memory

        023B 00     @cbnk   db      0               ; bank for processor operations


        023C                end
AUXIN           0198     99    397#
AUXIST          017D    113    367#
AUXOST          010C    114    277#
AUXOUT          00E0     98    230#
BANKED          FFFF      8#   186
BAUD110         0003
BAUD1200        0008
BAUD134         0004
BAUD150         0005
BAUD1800        0009
BAUD19200       000F
BAUD2400        000A
BAUD300         0006
BAUD3600        000B
BAUD4800        000C
BAUD50          0001
BAUD600         0007
BAUD7200        000D
BAUD75          0002
BAUD9600        000E
BAUDNONE        0000
BELL            0007     27#   632
BNKSEL          0225    124    475#
BOOT            0000     91    138#
BOOT1           0063    164    168#
BOOTSTACK       00D2    139    178    198#
CCP             0100     31#   171    181
CI1             016F    319    345#
CINEXT          019E    403#   411
CINITLOOP       0005    141#   143
CIRDY           01B2    408    415#
CISNEXT         0182    372#   379
CIST1           015D    318    331#   375    406
CONEXT          00EB    244#   258
CONIN           0192     95    388#
CONOST          0106    112    267#
CONOUT          00DA     96    220#
CONST           0177     94    357#
COSNEXT         0117    292#   304
COST1           0166    327    338#
COSTER          012C    250    297    308#
CR              000D     25#   632
CTLQ            0011     28#   320
CTLS            0013     29#   323
DEVTBL          00D2    115    204#
DINITLOOP       0017    149#   163
DINITNEXT       0036    152    161#
DRIVEMSG        00D1    463    632#
FALSE           0000      6#
FLUSH           00CF    120    626#
GETDRV          00D6    118    211#
HOME            006E    101    520#
INSCAN          019B    390    400#   413
IPCHL           01B6    159    423#   511
ISTSCAN         0180    359    370#
LF              000A     26#   632
LIST            00E6     97    239#
LISTST          0112    109    287#
MBINOUT         0003
MBINPUT         0001
MBOUTPUT        0002
MBSERIAL        0008
MBSOFTBAUD      0004
MBXONXOFF       0010    314
MULTIO          00CB    119    619#
```

**Listing E-1. (continued)**

```
NEXT             01D1   443#  456
NEXTDIGIT        01E6   452#
NOTFIRSTSELECT   006D   506   513#
NOTOUTDEVICE     00FF   246   255#
NOTOUTREADY      00F1   249#  250
NOTQ             0150   320   322#
NOTS             0157   323   325#
OSTSCAN          0115   269   279   290#
OUTSCAN          00E9   223   232   242#
PDECL            01D3   445#  447
PMSGEXIT         01C8   432   436#
PMSGLOOP         01B9   431#  435
READ             0094   106   585#
RWCOMMON         00BD   591   606#
SECTORMSG        00E8   467   634#
SECTRN           0089   110   573#
SELDSK           003F   102   500#
SETBNK           0085   125   562#
SETDMA           007D   105   550#
SETJUMPS         0078   169   179   184#
SETSEC           0077   104   538#
SETTRK           0071   103   528#
STOPLOOP         01DE   446   448#
TABLE10          01F3   442   459#
TRACKMSG         00E3   465   633#
TRUE             FFFF     5#    6     8
WBOOT            006C    92   177#
WRITE            00AA   107   599#
XOFFLIST         022B   317   481#
?AUXI            0015    79    99#
?AUXIS           0036    82   113#
?AUXO            0012    79    98#
?AUXOS           0039    82   114#
?BANK            0000    63   477
?BNKSL           0051    83   124#  187
?BOOT            0000    79    91#
?CI              0000    49   347   417
?CINIT           0000    50   116   142
?CIST            0000    49   333
?CO              0000    49   252
?CONIN           0009    79    95#
?CONO            000C    79    96#  434   450   464
?CONOS           0033    82   112#
?CONST           0006    79    94#
?COST            0000    49   316   340
?DEVIN           003F    82   116#
?DRTBL           0042    82   118#
?DVTBL           003C    82   115#
?FLUSH           0048    83   120#
?HOME            0018    80   101#
?INIT            0000    44   145
?LDCCP           0000    45   170
?LIST            000F    79    97#
?LISTS           002D    81   109#
?MLTIO           0045    83   119#
?MOV             004B    83   122#
?MOVE            0000    62   122
?PDEC            01CB    71   441#  466   468
?PDERR           01FD    72   462#
?PMSG            01B7    71   427#  463   465   467
?READ            0027    80   106#
?RLCCP           0000    45   180
?SCTRN           0030    81   110#
?SLDSK           001B    80   102#
?STBNK           0054    83   125#
?STDMA           0024    80   105#
?STSEC           0021    80   104#
?STTRK           001E    80   103#
?TIM             004E    83   123#
?TIME            0000    67   123
?WBOOT           0003    79    92#  192
?WRITE           002A    80   107#
?XMOV            0057    83   126#
?XMOVE           0000    62   126
@ADRV            00ED    56   156   464   501   586   600   639#
@AIVEC           0000    38   368   398
@AOVEC           0000    38   231   278
@BNKBF           0000    40
@CBNK            023B    61   476   554   650#
@CIVEC           0000    38   358   389
@CNT             00F5    57   620   644#
@COVEC           0000    38   222   268
```

**Listing E-1. (continued)**

```
@CTBL           0000   51   205   313
@DBNK           00F6   57   563   645#
@DMA            00F3   57   552   643#
@DTBL           0000   55   148   212   503   587   601
@LOVEC          0000   38   240   288
@MXTPA          0000   39   193
@RDRV           00EE   56   155   508   610   640#
@SECT           00F1   56   468   540   642#
@TRK            00EF   56   466   530   641#
```

**Listing E-1. (continued)**

End of Appendix E

# Appendix F
# System Control Block Definition for CP/M 3 BIOS

The SCB.ASM module contains the public definitions of the various fields in the System Control Block. The BIOS can reference the public variables.

```
                                title 'System Control Block Definition for CP/M3 BIOS'

                                public @civec, @covec, @aivec, @aovec, @lovec, @bnkbf
                                public @crdma, @crdsk, @vinfo, @resel, @fx, @usrcd
                                public @mltio, @ermde, @erdsk, @media, @bflgs
                                public @date, @hour, @min, @sec, ?erjmp, @mxtpa


FE00 =          scb$base equ    0FE00H          ; Base of the SCB

FE22 =          @CIVEC   equ    scb$base+22h    ; Console Input Redirection
                                                ; Vector (word, r/w)
FE24 =          @COVEC   equ    scb$base+24h    ; Console Output Redirection
                                                ; Vector (word, r/w)
FE26 =          @AIVEC   equ    scb$base+26h    ; Auxiliary Input Redirection
                                                ; Vector (word, r/w)
FE28 =          @AOVEC   equ    scb$base+28h    ; Auxiliary Output Redirection
                                                ; Vector (word, r/w)
FE2A =          @LOVEC   equ    scb$base+2Ah    ; List Output Redirection
                                                ; Vector (word, r/w)
FE35 =          @BNKBF   equ    scb$base+35h    ; Address of 128 Byte Buffer
                                                ; for Banked BIOS (word, r/o)
FE3C =          @CRDMA   equ    scb$base+3Ch    ; Current DMA Address
                                                ; (word, r/o)
FE3E =          @CRDSK   equ    scb$base+3Eh    ; Current Disk (byte, r/o)
FE3F =          @VINFO   equ    scb$base+3Fh    ; BDOS Variable "INFO"
                                                ; (word, r/o)
FE41 =          @RESEL   equ    scb$base+41h    ; FCB Flag (byte, r/o)
FE43 =          @FX      equ    scb$base+43h    ; BDOS Function for Error
                                                ; Messages (byte, r/o)
FE44 =          @USRCD   equ    scb$base+44h    ; Current User Code (byte, r/o)
FE4A =          @MLTIO   equ    scb$base+4Ah    ; Current Multi-Sector Count
                                                ; (byte,r/w)
FE4B =          @ERMDE   equ    scb$base+4Bh    ; BDOS Error Mode (byte, r/o)
FE51 =          @ERDSK   equ    scb$base+51h    ; BDOS Error Disk (byte,r/o)
FE54 =          @MEDIA   equ    scb$base+54h    ; Set by BIOS to indicate
                                                ; open door (byte,r/w)
FE57 =          @BFLGS   equ    scb$base+57h    ; BDOS Message Size Flag (byte,r/o)
FE58 =          @DATE    equ    scb$base+58h    ; Date in Days Since 1 Jan 78
                                                ; (word, r/w)
FE5A =          @HOUR    equ    scb$base+5Ah    ; Hour in BCD (byte, r/w)
FE5B =          @MIN     equ    scb$base+5Bh    ; Minute in BCD (byte, r/w)
FE5C =          @SEC     equ    scb$base+5Ch    ; Second in BCD (byte, r/w)
FE5F =          ?ERJMP   equ    scb$base+5Fh    ; BDOS Error Message Jump
                                                ; (word, r/w)
FE62 =          @MXTPA   equ    scb$base+62h    ; Top of User TPA
                                                ; (address at 6,7)(word, r/o)
0000                     end
```

**Listing F-1. System Control Block Definition for CP/M 3 BIOS**

```
SCBBASE      FE00    9#   11   13   15   17   19   21   23   25   26
                     28   29   31   32   34   35   36   38   39   41
                     42   43   44   46
?ERJMP       FE5F    6    44#
@AIVEC       FE26    3    15#
@AOVEC       FE28    3    17#
@BFLGS       FE57    5    38#
@BNKBF       FE35    3    21#
@CIVEC       FE22    3    11#
@COVEC       FE24    3    13#
@CRDMA       FE3C    4    23#
@CRDSK       FE3E    4    25#
@DATE        FE58    6    39#
@ERDSK       FE51    5    35#
@ERMDE       FE4B    5    34#
@FX          FE43    4    29#
@HOUR        FE5A    6    41#
@LOVEC       FE2A    3    19#
@MEDIA       FE54    5    36#
@MIN         FE5B    6    42#
@MLTIO       FE4A    5    32#
@MXTPA       FE62    6    46#
@RESEL       FE41    4    28#
@SEC         FE5C    6    43#
@USRCD       FE44    4    31#
@VINFO       FE3F    4    26#
```

**Listing F-1. (continued)**

End of Appendix F

# Appendix G
# Equates for Mode Byte Bit Fields

```
        ; equates for mode byte bit fields

mb$input            equ 0000$0001b  ; device may do input
mb$output           equ 0000$0010b  ; device may do output
mb$in$out           equ mb$input+mb$output
mb$soft$baud        equ 0000$0100b  ; software selectable baud rates
mb$serial           equ 0000$1000b  ; device may use protocol
mb$xon$xoff         equ 0001$0000b  ; XON/XOFF protocol enabled

baud$none           equ 0           ; no baud rate associated with device
baud$50             equ 1           ; 50 baud
baud$75             equ 2           ; 75 baud
baud$110            equ 3           ; 110 baud
baud$134            equ 4           ; 134.5 baud
baud$150            equ 5           ; 150 baud
baud$300            equ 6           ; 300 baud
baud$600            equ 7           ; 600 baud
baud$1200           equ 8           ; 1200 baud
baud$1800           equ 9           ; 1800 baud
baud$2400           equ 10          ; 2400 baud
baud$3600           equ 11          ; 3600 baud
baud$4800           equ 12          ; 4800 baud
baud$7200           equ 13          ; 7200 baud
baud$9600           equ 14          ; 9600 baud
baud$19200          equ 15          ; 19.2k baud
```

**Listing G-1. Equates for Mode Byte Fields: MODEBAUD.LIB**

End of Appendix G

# Appendix H
# Macro Definitions for CP/M 3 BIOS Data Structures

```
        Macro Definitions for CP/M3 BIOS Data Structures.

        ; dtbl  <dph0,dph1,...>          - drive table

        ; dph   translate$table,         - disk parameter header
        ;       disk$parameter$block,
        ;       checksum$size,                  (optional)
        ;       alloc$size                      (optional)

        ; skew  sectors,                 - skew table
        ;       skew$factor,
        ;       first$sector$number

        ; dpb   physical$sector$size,    - disk parameter block
        ;       physical$sectors$per$track,
        ;       number$tracks,
        ;       block$size,
        ;       number$dir$entries,
        ;       track$offset,
        ;       checksum$vec$size               (optional)

;       Drive Table.  Contains 16 one word entries.

dtbl macro ?list
    local ?n
?n  set 0
    irp ?drv,<?list>
?n  set ?n+1
        dw      ?drv
    endm

    if ?n > 16
.' Too many drives.  Max 16 allowed'
        exitm
    endif

    if ?n < 16
        rept (16-?n)
        dw      0
        endm
    endif
 endm

dph macro ?trans,?dpb,?csize,?asize
    local ?csv,?alv
        dw ?trans               ; translate table address
        db 0,0,0,0,0,0,0,0,0    ; BDOS Scratch area
        db 0                    ; media flag


        dw ?dpb                 ; disk parameter block
   if not nul ?csize
        dw ?csv                 ; checksum vector
   else
        dw OFFFEh               ; checksum vector allocated by GENCPM
   endif
   if not nul ?asize
        dw ?alv                 ; allocation vector
   else
        dw OFFFEh               ; alloc vector allocated by GENCPM
   endif
        dw 0fffeh,0fffeh,0fffeh ; dirbcb, dtabcb, hash alloc'd by GENCPM
        db 0                    ; hash bank
```

**Listing H-1. Macro Definitions for CP/M 3 BIOS Data Structures**

```
    if not nul ?csize
?csv    ds      ?csize          ; checksum vector
    endif
    if not nul ?asize
?alv    ds      ?asize          ; allocation vector
    endif

    endm

dpb macro ?psize,?pspt,?trks,?bls,?ndirs,?off,?ncks
    local ?spt,?bsh,?blm,?exm,?dsm,?drm,?al0,?al1,?cks,?psh,?psm
    local ?n
;; physical sector mask and physical sector shift
    ?psh        set 0
    ?n          set ?psize/128
    ?psm        set ?n-1
        rept 8
        ?n      set ?n/2
            if ?n = 0
            exitm
            endif
        ?psh    set ?psh + 1
        endm
    ?spt        set ?pspt*(?psize/128)

    ?bsh        set 3
    ?n          set ?bls/1024
        rept 8
        ?n      set ?n/2
            if ?n = 0
            exitm
            endif
        ?bsh    set ?bsh + 1
        endm
    ?blm        set ?bls/128-1
    ?size       set (?trks-?off)*?spt
    ?dsm        set ?size/(?bls/128)-1

    ?exm        set ?bls/1024
        if ?dsm > 255
            if ?bls = 1024

.'Error, can''t have this size disk with 1k block size'
            exitm
            endif
        ?exm    set ?exm/2
        endif
    ?exm        set ?exm-1
    ?all        set 0
    ?n          set (?ndirs*32+?bls-1)/?bls
        rept ?n
        ?all    set (?all shr 1) or 8000h
        endm
    ?al0        set high ?all
    ?al1        set low ?all
    ?drm        set ?ndirs-1
    if not nul ?ncks
        ?cks    set ?ncks
    else
        ?cks    set ?ndirs/4
    endif
        dw      ?spt            ; 128 byte records per track
        db      ?bsh,?blm       ; block shift and mask
        db      ?exm            ; extent mask
        dw      ?dsm            ; maximum block number
        dw      ?drm            ; maximum directory entry number
        db      ?al0,?al1       ; alloc vector for directory
        dw      ?cks            ; checksum size
        dw      ?off            ; offset for system tracks
        db      ?psh,?psm       ; physical sector size shift and mask
    endm
```

**Listing H-1. (continued)**

```
;
gcd macro ?m,?n
            ;;  greatest common divisor of m,n
                    ;;  produces value gcdn as result
                    ;;  (used in sector translate table generation)
    ?gcdm       set ?m  ;;variable for m
    ?gcdn       set ?n  ;;variable for n
    ?gcdr       set 0   ;;variable for r
        rept 65535
        ?gcdx   set ?gcdm/?gcdn
        ?gcdr   set ?gcdm - ?gcdx*?gcdn
            if ?gcdr = 0
            exitm
            endif
        ?gcdm   set ?gcdn
        ?gcdn   set ?gcdr
        endm
    endm

skew macro ?secs,?skf,?fsc
;;      generate the translate table
    ?nxtsec     set 0   ;;next sector to fill
    ?nxtbas     set 0   ;;moves by one on overflow
    gcd %?secs,?skf
    ;;  ?gcdn = gcd(?secs,skew)
    ?neltst     set ?secs/?gcdn
    ;;  neltst is number of elements to generate
    ;;  before we overlap previous elements
    ?nelts      set ?neltst     ;;counter
        rept ?secs      ;;once for each sector
        db      ?nxtsec+?fsc
        ?nxtsec set ?nxtsec+?skf
            if ?nxtsec >= ?secs
            ?nxtsec     set ?nxtsec-?secs
            endif
        ?nelts  set ?nelts-1
            if ?nelts = 0
            ?nxtbas     set ?nxtbas+1
            ?nxtsec     set ?nxtbas
            ?nelts      set ?neltst
            endif
        endm
    endm
```

**Listing H-1. (continued)**

End of Appendix H

# Appendix I
# ACS 8000-15 BIOS Modules

## I.1 Boot Loader Module for CP/M 3

The BOOT.ASM module performs system initialization other than character and disk I/O. BOOT loads the CCP for cold starts and reloads it for warm starts. Note that the device drivers in the Digital Research sample BIOS initialize devices for a polled, and not an interrupt-driven, environment.

```
                                title   'Boot loader module for CP/M 3.0'

FFFF =                  true equ -1
0000 =                  false equ not true

FFFF =                  banked  equ true

                                public  ?init,?ldccp,?rlccp,?time
                                extrn   ?pmsg,?conin
                                extrn   @civec,@covec,@aivec,@aovec,@lovec
                                extrn   @cbnk,?bnksl

                                maclib ports
                                maclib z80

0005 =                  bdos    equ 5

                                if banked
0001 =                  tpa$bank        equ 1
                                else
                        tpa$bank        equ 0
                                endif

                                dseg    ; init done from banked memory

                        ?init:
0000 2100802200                 lxi h,08000h ! shld @civec ! shld @covec        ; assign console to CRT:
0009 2100402200                 lxi h,04000h ! shld @lovec                      ; assign printer to LPT:
000F 2100202200                 lxi h,02000h ! shld @aivec ! shld @aovec        ; assign AUX to CRT1:
0018 21EF00CD25                 lxi h,init$table ! call out$blocks      ; set up misc hardware
001E 218700CD00                 lxi h,signon$msg ! call ?pmsg           ; print signon message
0024 C9                         ret

                        out$blocks:
0025 7EB7C847                   mov a,m ! ora a ! rz ! mov b,a
0029 234E23                     inx h ! mov c,m ! inx h
                                outir
002C+EDB3                       DB      0EDH,0B3H
002E C32500                     jmp out$blocks


                                cseg    ; boot loading most be done from resident memory

                            ;   This version of the boot loader loads the CCP from a file
                            ;   called CCP.COM on the system drive (A:).


                        ?ldccp:
                                ; First time, load the A:CCP.COM file into TPA
0000 AF32DB00                   xra a ! sta ccp$fcb+15          ; zero extent
0004 21000022EC                 lxi h,0 ! shld fcb$nr           ; start at beginning of file
000A 11CC00CD73                 lxi d,ccp$fcb ! call open       ; open file containing CCP
0010 3CCA4A00                   inr a ! jz no$CCP               ; error if no file...
0014 110001CD78                 lxi d,0100h ! call setdma       ; start of TPA
001A 118000CD7D                 lxi d,128 ! call setmulti       ; allow up to 16k bytes
0020 11CC00CD82                 lxi d,ccp$fcb ! call read       ; load the thing
                                                                ; now,
```

**Listing I-1. Boot Loader Module for CP/M 3**

```
                                                                ;   copy CCP to bank 0 for reloading
0026 2100010180              lxi h,0100h ! lxi b,0C80h          ; clone 3.125K, just in case
002C 3A0000F5                lda @cbnk ! push psw               ; save current bank
                      ld$1:
0030 3E01CD0000              mvi a,tpa$bank ! call ?bnksl       ; select TPA
0035 7EF5                    mov a,m ! push psw                 ; get a byte
0037 3E02CD0000              mvi a,2 ! call ?bnksl              ; select extra bank
003C F177                    pop psw ! mov m,a                  ; save the byte
003E 230B                    inx h ! dcx b                      ; bump pointer, drop count
0040 78B1                    mov a,b ! ora c                    ; test for done
0042 C23000                  jnz ld$1
0045 F1CD0000                pop psw ! call ?bnksl              ; restore original bank
0049 C9                      ret

                      no$CCP:                   ; here if we couldn't find the file
004A 21AB00CD00              lxi h,ccp$msg ! call ?pmsg         ; report this...
0050 CD0000                  call ?conin                        ; get a response
0053 C30000                  jmp ?ldccp                         ; and try again


                      ?rlccp:
0056 2100010180              lxi h,0100h ! lxi b,0C80h          ; clone 3.125K
                      rl$1:
005C 3E02CD0000              mvi a,2 ! call ?bnksl              ; select extra bank
0061 7EF5                    mov a,m ! push psw                 ; get a byte
0063 3E01CD0000              mvi a,tpa$bank ! call ?bnksl       ; select TPA
0068 F177                    pop psw ! mov m,a                  ; save the byte
006A 230B                    inx h ! dcx b                      ; bump pointer, drop count
006C 78B1                    mov a,b ! ora c                    ; test for done
006E C25C00                  jnz rl$1
0071 C9                      ret

                          ; No external clock.
                      ?time:
0072 C9                      ret

                             ; CP/M BDOS Function Interfaces

                      open:
0073 0E0FC30500              mvi c,15 ! jmp bdos                ; open file control block

                      setdma:
0078 0E1AC30500              mvi c,26 ! jmp bdos                ; set data transfer address

                      setmulti:
007D 0E2CC30500              mvi c,44 ! jmp bdos                ; set record count

                      read:
0082 0E14C30500              mvi c,20 ! jmp bdos                ; read records


0087 0D0A0D0A43signon$msg        db      13,10,13,10,'CP/M Version 3.0, sample BIOS',13,10,0

00AB 0D0A42494Fccp$msg           db      13,10,'BIOS Err on A: No CCP.COM file',0


00CC 0143435020ccp$fcb           db      1,'CCP     ','COM',0,0,0,0
00DC                             ds      16
00EC 000000     fcb$nr           db      0,0,0

00EF 0326CFFF07init$table        db      3,p$zpio$3a,0CFh,0FFh,07h      ; set up config port
00F4 0327CF0007                  db      3,p$zpio$3b,0CFh,000h,07h      ; set up bank port
00F9 012500                      db      1,p$bank$select,0      ; select bank 0
00FC 00                          db      0                      ; end of init$table

00FD                         end
BANKED             FFFF    6#   18
BC                 0000
BDOS               0005   16#   97   100   103   106
CCPFCB             00CC   50    52    56   114#
CCPMSG             00AB   73   111#
DE                 0002
FALSE              0000    4#
FCBNR              00EC   51   116#
HL                 0004
INITTABLE          00EF   30   118#
IX                 0004
IY                 0004
LD1                0030   61#   68
NOCCP              004A   53    72#
```

**Listing I-1. (continued)**

```
OPEN            0073   52   96#
OUTBLOCKS       0025   30   34#   39
PBANKSELECT     0025  120
PBAUDCON1       000C
PBAUDCON2       0030
PBAUDCON34      0031
PBAUDLPT1       000E
PBAUDLPT2       0032
PBOOT           0014
PCENTDATA       0011
PCENTSTAT       0010
PCON2DATA       002C
PCON2STAT       002D
PCON3DATA       002E
PCON3STAT       002F
PCON4DATA       002A
PCON4STAT       002B
PCONFIGURATION  0024
PCRTDATA        001C
PCRTSTAT        001D
PFDCMND         0004
PFDDATA         0007
PFDINT          0008
PFDMISC         0009
PFDSECTOR       0006
PFDSTAT         0004
PFDTRACK        0005
PINDEX          000F
PLPT2DATA       0028
PLPT2STAT       0029
PLPTDATA        001E
PLPTSTAT        001F
PRTC            0033
PSELECT         0008
PWD1797         0004
PZCTC1          000C
PZCTC2          0030
PZDART          001C
PZDMA           0000
PZPIO1          0008
PZPIO1A         000A
PZPIO1B         000B
PZPIO2          0010
PZPIO2A         0012
PZPIO2B         0013
PZPIO3          0024
PZPIO3A         0026  118
PZPIO3B         0027  119
PZSIO1          0028
PZSIO2          002C
READ            0082   56  105#
RL1             005C   80#   87
SETDMA          0078   54   99#
SETMULTI        007D   55  102#
SIGNONMSG       0087   31  109#
TPABANK         0001   19#   21#   62    83
TRUE            FFFF    3#    4     6
?BNKSL          0000   11    62    64    69    81    83
?CONIN          0000    9    74
?INIT           0000    8    26#
?LDCCP          0000    8    48#   75
?PMSG           0000    9    31    73
?RLCCP          0056    8    78#
?TIME           0072    8    91#
@AIVEC          0000   10    29
@AOVEC          0000   10    29
@CBNK           0000   11    60
@CIVEC          0000   10    27
@COVEC          0000   10    27
@LOVEC          0000   10    28
```

**Listing I-1. (continued)**

## I.2 Character I/O Handler for Z80 Chip-based System

The CHARIO.ASM module performs all character device initialization, input, output, and status polling. CHARIO contains the character device characteristics table.

```
                                title 'Character I/O handler for z80 chip based system'

                        ; Character I/O for the Modular CP/M 3 BIOS

                                ; limitations:

                                ;               baud rates 19200,7200,3600,1800 and 134
                                ;                       are approximations.

                                ;               9600 is the maximum baud rate that is likely
                                ;                       to work.

                                ;               baud rates 50, 75, and 110 are not supported


                                public  ?cinit,?ci,?co,?cist,?cost
                                public  @ctbl

                                maclib Z80      ; define Z80 op codes
                                maclib ports    ; define port addresses
                                maclib modebaud ; define mode bits and baud equates

0006 =                  max$devices     equ 6

                                cseg

                        ?cinit:
0000 79FE06CA42                 mov a,c ! cpi max$devices ! jz cent$init ; init parallel printer
0006 D0                         rnc                                     ; invalid device
0007 692600                     mov l,c ! mvi h,0                       ; make 16 bits from device number
000A E5                         push h                                  ; save device in stack
000B 292929                     dad h ! dad h ! dad h                   ; *8
000E 11E900196E                 lxi d,@ctbl+7 ! dad d ! mov l,m ; get baud rate
0013 7DFE07                     mov a,l ! cpi baud$600                  ; see if baud > 300
0016 3E44D21D00                 mvi a,44h ! jnc hi$speed                ; if >= 600, use *16 mode
001B 3EC4                       mvi a,0C4h                              ;    else, use *64 mode
                        hi$speed:
001D 323501                     sta sio$reg$4
0020 2600111B01                 mvi h,0 ! lxi d,speed$table ! dad d     ; point to counter entry
0026 7E322E01                   mov a,m ! sta speed                     ; get and save ctc count
002A E1                         pop h                                   ; recover
002B 11DC0019                   lxi d,data$ports ! dad d                ; point at SIO port address
002F 7E3C323001                 mov a,m ! inr a ! sta sio$port  ; get and save port
0034 11FAFF19                   lxi d,baud$ports-data$ports ! dad d     ; offset to baud rate port
0038 7E322C01                   mov a,m ! sta ctc$port                  ; get and save
003C 212B01                     lxi h,serial$init$tbl
003F C34500                     jmp stream$out

                        cent$init:
0042 213901                     lxi h,pio$init$tbl

                        stream$out:
0045 7EB7C8                     mov a,m ! ora a ! rz
0048 47234E23                   mov b,a ! inx h ! mov c,m ! inx h
                                outir
004C+EDB3                       DB      0EDH,0B3H
004E C34500                     jmp stream$out


                        ?ci:                    ; character input

0051 78FE06D263                 mov a,b ! cpi 6 ! jnc null$input ; can't read from centronics
                        cil:
0057 CD6600CA57                 call ?cist ! jz cil                     ; wait for character ready
005D 0D                         dcr c ! inp a                           ; get data
005E+ED78                       DB      0EDH,A*8+40H
0060 E67F                       ani 7Fh                                 ; mask parity
0062 C9                         ret
```

**Listing I-2. Character I/O Handler for Z80 Chip-based System**

```
                        null$input:
0063 3E1A                       mvi a,1Ah                       ; return a ctl-Z for no device
0065 C9                         ret

                        ?cist:                  ; character input status

0066 78FE06D27D                 mov a,b ! cpi 6 ! jnc null$status ; can't read from centronics
006C 682600                     mov l,b ! mvi h,0               ; make device number 16 bits
006F 11DC0019                   lxi d,data$ports ! dad d        ; make pointer to port address
0073 4E0C                       mov c,m ! inr c                 ; get SIO status port
                                inp a                           ; read from status port
0075+ED78                       DB      0EDH,A*8+40H
0077 E601                       ani 1                   ; isolate RxRdy
0079 C8                         rz                              ; return with zero
007A F6FF                       ori 0FFh
007C C9                         ret

                        null$status:
007D AFC9                       xra a ! ret

                        ?co:                    ; character output
007F 78FE06CA9E                 mov a,b ! cpi 6 ! jz centronics$out
0085 D29D00                     jnc null$output
0088 79F5                       mov a,c ! push psw              ; save character from <C>
008A C5                         push b                          ; save device number
                        co$spin:
008B CDB300CA8B                 call ?cost ! jz co$spin         ; wait for TxEmpty
0091 E16C2600                   pop h ! mov l,h ! mvi h,0       ; get device number in <HL>
0095 11DC0019                   lxi d,data$ports ! dad d        ; make address of port address
0099 4E                         mov c,m                         ; get port address
009A F1                         pop psw ! outp a                ; send data
009B+ED79                       DB      0EDH,A*8+41H
                        null$output:
009D C9                         ret

                        centronics$out:
009E DB10E620C2                 in p$centstat ! ani 20h ! jnz centronics$out
00A5 79D311                     mov a,c ! out p$centdata        ; give printer data
00A8 DB10F601D3                 in p$centstat ! ori 1 ! out p$centstat  ; set strobe
00AE E67ED310                   ani 7Eh ! out p$centstat                ; clear strobe
00B2 C9                         ret

                        ?cost:                  ; character output status
00B3 78FE06CACD                 mov a,b ! cpi 6 ! jz cent$stat
00B9 D27D00                     jnc null$status
00BC 682600                     mov l,b ! mvi h,0
00BF 11DC0019                   lxi d,data$ports ! dad d
00C3 4E0C                       mov c,m ! inr c
                                inp a                           ; get input status
00C5+ED78                       DB      0EDH,A*8+40H
00C7 E604C8                     ani 4 ! rz                      ; test transmitter empty
00CA F6FFC9                     ori 0FFh ! ret                  ; return true if ready


                        cent$stat:
00CD DB102F                     in p$centstat ! cma
00D0 E620C8                     ani 20h ! rz
00D3 F6FFC9                     ori 0FFh ! ret

                        baud$ports:             ; CTC ports by physical device number
00D6 0C0E3031                   db      p$baud$con1,p$baud$lpt1,p$baud$con2,p$baud$con34
00DA 3132                       db      p$baud$con34,p$baud$lpt2

                        data$ports:             ; serial base ports by physical device number
00DC 1C1E2C2E                   db      p$crt$data,p$lpt$data,p$con2data,p$con3data
00E0 2A28                       db      p$con4data,p$lpt2data


00E2 4352542020@ctbl            db 'CRT   '     ; device 0, CRT port 0
00E8 0F                         db mb$in$out+mb$serial+mb$softbaud
00E9 0E                         db baud$9600
00EA 4C50542020                 db 'LPT   '     ; device 1, LPT port 0
00F0 1F                         db mb$in$out+mb$serial+mb$softbaud+mb$xonxoff
00F1 0E                         db baud$9600
00F2 4352543120                 db 'CRT1  '     ; device 2, CRT port 1
00F8 0F                         db mb$in$out+mb$serial+mb$softbaud
00F9 0E                         db baud$9600
00FA 4352543220                 db 'CRT2  '     ; device 3, CRT port 2
0100 0F                         db mb$in$out+mb$serial+mb$softbaud
0101 0E                         db baud$9600
```

**Listing I-2. (continued)**

```
 150   0102 4352543320        db 'CRT3  '     ; device 4, CRT port 3
 151   0108 0F                db mb$in$out+mb$serial+mb$softbaud
 152   0109 0E                db baud$9600
 153   010A 5641582020        db 'VAX   '     ; device 5, LPT port 1 used for VAX interface
 154   0110 0F                db mb$in$out+mb$serial+mb$softbaud
 155   0111 0E                db baud$9600
 156   0112 43454E2020        db 'CEN   '     ; device 6, Centronics parallel printer
 157   0118 02                db mb$output
 158   0119 00                db baud$none
 159   011A 00                db 0                    ; table terminator
 160
 161
 162   011B 00FFFFFFE9speed$table     db     0,255,255,255,233,208,104,208,104,69,52,35,26,17,13,7
 163
 164                  serial$init$tbl
 165   012B 02                    db 2                 ; two bytes to CTC
 166   012C           ctc$port    ds 1                 ; port address of CTC
 167   012D 47                    db 47h               ; CTC mode byte
 168   012E           speed       ds 1                 ; baud multiplier
 169   012F 07                    db 7                 ; 7 bytes to SIO
 170   0130           sio$port    ds 1                 ; port address of SIO
 171   0131 1803E104              db 18h,3,0E1h,4
 172   0135           sio$reg$4   ds 1
 173   0136 05EA                  db 5,0EAh
 174   0138 00                    db 0                 ; terminator
 175
 176   0139 02130F07  pio$init$tbl    db     2,p$zpio$2b,0Fh,07h
 177   013D 0312CFF807                db     3,p$zpio$2a,0CFh,0F8h,07h
 178   0142 00                        db 0
 179
 180   0143                   end
BAUD110          0003
BAUD1200         0008
BAUD134          0004
BAUD150          0005
BAUD1800         0009
BAUD19200        000F
BAUD2400         000A
BAUD300          0006
BAUD3600         000B
BAUD4800         000C
BAUD50           0001
BAUD600          0007     34
BAUD7200         000D
BAUD75           0002
BAUD9600         000E    140    143    146    149    152    155
BAUDNONE         0000    158
BAUDPORTS        00D6     44    129#
BC               0000
CENTINIT         0042     28     49#
CENTRONICSOUT    009E     91    105#   106
CENTSTAT         00CD    113    124#
CI1              0057     63#    64
COSPIN           008B     95#    96
CTCPORT          012C     45    166#
DATAPORTS        00DC     42     44     78     98    116    133#
DE               0002
HISPEED          001D     35     37#
HL               0004
IX               0004
IY               0004
MAXDEVICES       0006     23#    28
MBINOUT          0003    139    142    145    148    151    154
MBINPUT          0001
MBOUTPUT         0002    157
MBSERIAL         0008    139    142    145    148    151    154
MBSOFTBAUD       0004    139    142    145    148    151    154
MBXONXOFF        0010    142
NULLINPUT        0063     62     70#
NULLOUTPUT       009D     92    102#
NULLSTATUS       007D     76     87#   114
PBANKSELECT      0025
PBAUDCON1        000C    130
PBAUDCON2        0030    130
PBAUDCON34       0031    130    131
PBAUDLPT1        000E    130
PBAUDLPT2        0032    131
PBOOT            0014
PCENTDATA        0011    107
PCENTSTAT        0010    106    108    108    109    125
PCON2DATA        002C    134
```

**Listing I-2. (continued)**

```
PCON2STAT        002D
PCON3DATA        002E   134
PCON3STAT        002F
PCON4DATA        002A   135
PCON4STAT        002B
PCONFIGURATION   0024
PCRTDATA         001C   134
PCRTSTAT         001D
PFDCMND          0004
PFDDATA          0007
PFDINT           0008
PFDMISC          0009
PFDSECTOR        0006
PFDSTAT          0004
PFDTRACK         0005
PINDEX           000F
PIOINITTBL       0139    50    176#
PLPT2DATA        0028   135
PLPT2STAT        0029
PLPTDATA         001E   134
PLPTSTAT         001F
PRTC             0033
PSELECT          0008
PWD1797          0004
PZCTC1           000C
PZCTC2           0030
PZDART           001C
PZDMA            0000
PZPIO1           0008
PZPIO1A          000A
PZPIO1B          000B
PZPIO2           0010
PZPIO2A          0012   177
PZPIO2B          0013   176
PZPIO3           0024
PZPIO3A          0026
PZPIO3B          0027
PZSIO1           0028
PZSIO2           002C
SERIALINITTBL    012B    46    164#
SIOPORT          0130    43    170#
SIOREG4          0135    38    172#
SPEED            012E    40    168#
SPEEDTABLE       011B    39    162#
STREAMOUT        0045    47     52#    57
?CI              0051    16     60#
?CINIT           0000    16     27#
?CIST            0066    16     64     74#
?CO              007F    16     90#
?COST            00B3    16     96    112#
@CTBL            00E2    17     33    138#
```

**Listing I-2. (continued)**

## I.3 Drive Table

The DRVTBL.ASM module points to the data structures for each configured disk drive. The drive table determines which physical disk unit is associated with which logical drive. The data structure for each disk drive is called an Extended Disk Parameter Header (XDPH).

```
                                   public @dtbl
                                   extrn fdsd0,fdsd1

                                   cseg

     0000 00000000  @dtbl          dw fdsd0,fdsd1
     0004 0000000000               dw 0,0,0,0,0,0,0,0,0,0,0,0,0,0        ; drives C-P non-existent

     0020                          end
FDSD0          0000       2        6
FDSD1          0000       2        6
@DTBL          0000       1        6#
```

**Listing I-3. Drive Table**

## I.4 Z80 DMA Single-density Disk Handler

The FD1797SD module initializes the disk controllers for the disks described in the Disk Parameter Headers and Disk Parameter Blocks contained in this module. FD1797SD is written for hardware that supports Direct Memory Access (DMA).

```
        title 'wd1797 w/ Z80 DMA Single density diskette handler'

;    CP/M-80 Version 3     --  Modular BIOS

;       Disk I/O Module for wd1797 based diskette systems

        ;       Initial version 0.01,
        ;               Single density floppy only.     - jrp, 4 Aug 82

        dseg

    ; Disk drive dispatching tables for linked BIOS

        public  fdsd0,fdsd1

    ; Variables containing parameters passed by BDOS

        extrn   @adrv,@rdrv
        extrn   @dma,@trk,@sect
        extrn   @dbnk

    ; System Control Block variables

        extrn   @ermde          ; BDOS error mode

    ; Utility routines in standard BIOS
```

**Listing I-4. Z80 DMA Single-density Disk Handler**

```

                                extrn   ?wboot  ; warm boot vector
                                extrn   ?pmsg   ; print message @<HL> up to 00, saves <BC> & <DE>
                                extrn   ?pdec   ; print binary number in <A> from 0 to 99.
                                extrn   ?pderr  ; print BIOS disk error header
                                extrn   ?conin,?cono     ; con in and out
                                extrn   ?const           ; get console status


                        ; Port Address Equates

                                maclib ports

                        ; CP/M 3 Disk definition macros

                                maclib cpm3

                        ; Z80 macro library instruction definitions

                                maclib z80

                        ; common control characters

 000D =             cr          equ 13
 000A =             lf          equ 10
 0007 =             bell        equ 7


                        ; Extended Disk Parameter Headers (XPDHs)

 0000 E600                      dw      fd$write
 0002 DC00                      dw      fd$read
 0004 DB00                      dw      fd$login
 0006 BE00                      dw      fd$init0
 0008 0000                      db      0,0              ; relative drive zero
                    fdsd0       dph     trans,dpbsd,16,31
 000A+A400                      DW TRANS                 ; TRANSLATE TABLE ADDRESS
 000C+0000000000                DB 0,0,0,0,0,0,0,0,0     ; BDOS SCRATCH AREA
 0015+00                        DB 0                     ; MEDIA FLAG
 0016+0000                      DW DPBSD                         ; DISK PARAMETER BLOCK
 0018+2300                      DW ??0001                        ; CHECKSUM VECTOR
 001A+3300                      DW ??0002                        ; ALLOCATION VECTOR
 001C+FEFFFEFFFE                DW 0FFFEH,0FFFEH,0FFFEH ; DIRBCB, DTABCB, HASH ALLOC'D BY GENCPM
 0022+00                        DB 0                     ; HASH BANK
 0023+              ??0001      DS      16               ; CHECKSUM VECTOR
 0033+              ??0002      DS      31               ; ALLOCATION VECTOR

 0052 E600                      dw      fd$write
 0054 DC00                      dw      fd$read
 0056 DB00                      dw      fd$login
 0058 CD00                      dw      fd$init1
 005A 0100                      db      1,0              ; relative drive one
                    fdsd1       dph     trans,dpbsd,16,31
 005C+A400                      DW TRANS                 ; TRANSLATE TABLE ADDRESS
 005E+0000000000                DB 0,0,0,0,0,0,0,0,0     ; BDOS SCRATCH AREA
 0067+00                        DB 0                     ; MEDIA FLAG
 0068+0000                      DW DPBSD                         ; DISK PARAMETER BLOCK
 006A+7500                      DW ??0003                        ; CHECKSUM VECTOR
 006C+8500                      DW ??0004                        ; ALLOCATION VECTOR
 006E+FEFFFEFFFE                DW 0FFFEH,0FFFEH,0FFFEH ; DIRBCB, DTABCB, HASH ALLOC'D BY GENCPM
 0074+00                        DB 0                     ; HASH BANK
 0075+              ??0003      DS      16               ; CHECKSUM VECTOR
 0085+              ??0004      DS      31               ; ALLOCATION VECTOR

                                cseg    ; DPB must be resident

                    dpbsd       dpb 128,26,77,1024,64,2
 0000+1A00                      DW      ??0005           ; 128 BYTE RECORDS PER TRACK
 0002+0307                      DB      ??0006,??0007    ; BLOCK SHIFT AND MASK
 0004+00                        DB      ??0008           ; EXTENT MASK
 0005+F200                      DW      ??0009           ; MAXIMUM BLOCK NUMBER
 0007+3F00                      DW      ??0010           ; MAXIMUM DIRECTORY ENTRY NUMBER
 0009+C000                      DB      ??0011,??0012    ; ALLOC VECTOR FOR DIRECTORY
 000B+1000                      DW      ??0013           ; CHECKSUM SIZE
 000D+0200                      DW      2                ; OFFSET FOR SYSTEM TRACKS
 000F+0000                      DB      ??0014,??0015    ; PHYSICAL SECTOR SIZE SHIFT AND MASK

                                dseg    ; rest is banked
```

**Listing I-4. (continued)**

```
                    trans   skew 26,6,1
00A4+01                     DB      ?NXTSEC+1
00A5+07                     DB      ?NXTSEC+1
00A6+0D                     DB      ?NXTSEC+1
00A7+13                     DB      ?NXTSEC+1
00A8+19                     DB      ?NXTSEC+1
00A9+05                     DB      ?NXTSEC+1
00AA+0B                     DB      ?NXTSEC+1
00AB+11                     DB      ?NXTSEC+1
00AC+17                     DB      ?NXTSEC+1
00AD+03                     DB      ?NXTSEC+1
00AE+09                     DB      ?NXTSEC+1
00AF+0F                     DB      ?NXTSEC+1
00B0+15                     DB      ?NXTSEC+1
00B1+02                     DB      ?NXTSEC+1
00B2+08                     DB      ?NXTSEC+1
00B3+0E                     DB      ?NXTSEC+1
00B4+14                     DB      ?NXTSEC+1
00B5+1A                     DB      ?NXTSEC+1
00B6+06                     DB      ?NXTSEC+1
00B7+0C                     DB      ?NXTSEC+1
00B8+12                     DB      ?NXTSEC+1
00B9+18                     DB      ?NXTSEC+1
00BA+04                     DB      ?NXTSEC+1
00BB+0A                     DB      ?NXTSEC+1
00BC+10                     DB      ?NXTSEC+1
00BD+16                     DB      ?NXTSEC+1


                        ; Disk I/O routines for standardized BIOS interface

                    ; Initialization entry point.

                    ;                   called for first time initialization.


                    fd$init0:
00BE 21CE00                 lxi h,init$table
                    fd$init$next:
00C1 7EB7C8                 mov a,m ! ora a ! rz
00C4 47234E23               mov b,a ! inx h ! mov c,m ! inx h
                            outir
00C8+EDB3                   DB      0EDH,0B3H
00CA C3C100                 jmp fd$init$next

                    fd$init1:       ; all initialization done by drive 0
00CD C9                     ret

00CE 040A           init$table      db 4,p$zpio$1A
00D0 CFC217FF                       db      11001111b, 11000010b, 00010111b,11111111b
00D4 040B                           db 4,p$zpio$1B
00D6 CFDD17FF                       db      11001111b, 11011101b, 00010111b,11111111b
00DA 00                             db 0


                    fd$login:
                                    ; This entry is called when a logical drive is about to
                                    ; be logged into for the purpose of density determination.

                                    ; It may adjust the parameters contained in the disk
                                    ; parameter header pointed at by <DE>

00DB C9                     ret     ; we have nothing to do in
                                    ;       simple single density only environment.


                    ; disk READ and WRITE entry points.

                                    ; these entries are called with the following arguments:

                                            ; relative drive number in @rdrv (8 bits)
                                            ; absolute drive number in @adrv (8 bits)
                                            ; disk transfer address in @dma (16 bits)
                                            ; disk transfer bank    in @dbnk (8 bits)
                                            ; disk track address    in @trk (16 bits)
                                            ; disk sector address   in @sect (16 bits)
                                            ; pointer to XDPH in <DE>
```

**Listing I-4. (continued)**

```
                                          ; they transfer the appropriate data, perform retries
                                          ; if necessary, then return an error code in <A>

                        fd$read:
00DC 211802                    lxi h,read$msg            ; point at " Read "
00DF 3E880601                  mvi a,88h ! mvi b,01h     ; 1797 read + Z80DMA direction
00E3 C3ED00                    jmp rw$common

                        fd$write:
00E6 211F02                    lxi h,write$msg           ; point at " Write "
00E9 3EA80605                  mvi a,0A8h ! mvi b,05h    ; 1797 write + Z80DMA direction
                           ;   jmp wr$common

                        rw$common:                       ; seek to correct track (if necessary),
                                                         ;       initialize DMA controller,
                                                         ;       and issue 1797 command.

00ED 222702                    shld operation$name                ; save message for errors
00F0 321102                    sta disk$command                   ; save 1797 command
00F3 7832A802                  mov a,b ! sta zdma$direction       ; save Z80DMA direction code
00F7 2A0000229F                lhld @dma ! shld  zdma$dma         ; get and save DMA address
00FD 3A00006F26                lda @rdrv ! mov l,a ! mvi h,0      ; get controller-relative disk drive
0103 11160219                  lxi d,select$table ! dad d         ; point to select mask for drive
0107 7E321202                  mov a,m ! sta select$mask          ; get select mask and save it
010B D308                      out p$select                       ; select drive
                        more$retries:
010D 0E0A                      mvi c,10                           ; allow 10 retries
                        retry$operation:
010F C5                        push b                             ; save retry counter

0110 3A12022113                lda select$mask ! lxi h,old$select ! cmp m
0117 77                        mov m,a
0118 C22D01                    jnz new$track                      ; if not same drive as last, seek

011B 3A00002114                lda @trk ! lxi h,old$track ! cmp m
0122 77                        mov m,a
0123 C22D01                    jnz new$track                      ; if not same track, then seek

0126 DB09E602C2                in p$fdmisc ! ani 2 ! jnz same$track    ; head still loaded, we are OK

                        new$track:      ; or drive or unloaded head means we should . . .
012D CDA901                    call check$seek         ; . . read address and seek if wrong track

0130 011B41                    lxi b,16667             ; 100 ms / (24 t states*250 ns)
                        spin$loop:                     ; wait for head/seek settling
0133 0B                        dcx b
0134 78B1                      mov a,b ! ora c
0136 C23301                    jnz spin$loop

                        same$track:
0139 3A0000D305                lda @trk ! out p$fdtrack           ; give 1797 track
013E 3A0000D306                lda @sect ! out p$fdsector         ;        and sector

0143 219A02                    lxi h,dma$block                    ; point to dma command block
0146 010011                    lxi b,dmab$length*256 + p$zdma     ; command block length and port address
                               outir                              ; send commands to Z80 DMA
0149+EDB3                      DB      0EDH,0B3H

014B DB25                      in p$bankselect                    ; get old value of bank select port
014D E63F47                    ani 3Fh ! mov b,a                  ; mask off DMA bank and save
0150 3A00000F0F                lda @dbnk ! rrc ! rrc              ; get DMA bank to 2 hi-order bits
0155 E6C0B0                    ani 0C0h ! ora b                   ; merge with other bank stuff
0158 D325                      out p$bankselect                   ; and select the correct DMA bank

015A 3A1102                    lda disk$command        ; get 1797 command
015D CDD501                    call exec$command       ; start it then wait for IREQ and read status
0160 321502                    sta disk$status         ; save status for error messages

0163 C1                        pop b                   ; recover retry counter
0164 B7C8                      ora a ! rz              ; check status and return to BDOS if no error

0166 E610                      ani 0001$0000b          ; see if record not found error
0168 C4A901                    cnz check$seek          ; if a record not found, we might need to seek

016B 0DC20F01                  dcr c ! jnz retry$operation

                           ; suppress error message if BDOS is returning errors to application...

016F 3A0000FEFF                lda @ermde ! cpi 0FFh ! jz hard$error
```

**Listing I-4. (continued)**

```
                                ; Had permanent error, print message like:

                                        ; BIOS Err on d: T-nn, S-mm, <operation> <type>, Retry ?

 0177 CD0000            call ?pderr             ; print message header

 017A 2A2702CD00        lhld operation$name ! call ?pmsg                ; last function

                                        ; then, messages for all indicated error bits

 0180 3A1502            lda disk$status         ; get status byte from last error
 0183 212902            lxi h,error$table       ; point at table of message addresses
                errml:
 0186 5E235623          mov e,m ! inx h ! mov d,m ! inx h ; get next message address
 018A 87F5              add a ! push psw        ; shift left and push residual bits with status
 018C EBDC0000EB        xchg ! cc ?pmsg ! xchg  ; print message, saving table pointer
 0191 F1C28601          pop psw ! jnz errml     ; if any more bits left, continue

 0195 218A02CD00        lxi h,error$msg ! call ?pmsg    ; print "<BEL>, Retry (Y/N) ? "
 019B CDF501            call u$conin$echo       ; get operator response
 019E FE59CA0D01        cpi 'Y' ! jz more$retries ; Yes, then retry 10 more times
                hard$error:                     ; otherwise,
 01A3 3E01C9            mvi a,1 ! ret           ;       return hard error to BDOS

                cancel:                         ; here to abort job
 01A6 C30000            jmp ?wboot              ; leap directly to warmstart vector


                                        ; subroutine to seek if on wrong track
                                        ; called both to set up new track or drive

                check$seek:
 01A9 C5                push b                          ; save error counter
 01AA CDE101            call read$id                    ; try to read ID, put track in <B>
 01AD CABE01            jz id$ok                        ; if OK, we're OK
 01B0 CDCE01            call step$out                   ; else step towards Trk 0
 01B3 CDE101            call read$id                    ; and try again
 01B6 CABE01            jz id$ok                        ; if OK, we're OK
 01B9 CDD301            call restore                    ; else, restore the drive
 01BC 0600              mvi b,0                         ; and make like we are at track 0
                id$ok:
 01BE 78D305            mov a,b ! out p$fdtrack         ; send current track to track port
 01C1 3A0000B8C1        lda @trk ! cmp b ! pop b ! rz   ; if its desired track, we are done
 01C7 D307              out p$fddata                    ; else, desired track to data port
 01C9 3E1A              mvi a,00011010b                 ; seek w/ 10 ms. steps
 01CB C3D501            jmp exec$command



                step$out:
 01CE 3E6A              mvi a,01101010b                 ; step out once at 10 ms.
 01D0 C3D501            jmp exec$command

                restore:
 01D3 3E0B              mvi a,00001011b                 ; restore at 15 ms
                      ; jmp exec$command


                exec$command:           ; issue 1797 command, and wait for IREQ
                                        ;       return status
 01D5 D304              out p$fdcmnd                    ; send 1797 command
                wait$IREQ:                              ; spin til IREQ
 01D7 DB08E640CA        in p$fdint ! ani 40h ! jz wait$IREQ
 01DE DB04              in p$fdstat                     ; get 1797 status and clear IREQ
 01E0 C9                ret

                read$id:
 01E1 21AB02            lxi h,read$id$block     ; set up DMA controller
 01E4 01000F            lxi b,length$id$dmab*256 + p$zdma ; for READ ADDRESS operation
                        outir
 01E7+EDB3              DB      0EDH,0B3H
 01E9 3EC4              mvi a,11000100b         ; issue 1797 read address command
 01EB CDD501            call exec$command       ; wait for IREQ and read status
 01EE E69D              ani 10011101b           ; mask status
 01F0 21110046          lxi h,id$buffer ! mov b,m       ; get actual track number in <B>
 01F4 C9                ret                     ; and return with Z flag true for OK

```

**Listing I-4. (continued)**

```
                        u$conin$echo:    ; get console input, echo it, and shift to upper case
0lF5 CD0000B7CA                 call ?const ! ora a ! jz u$cl    ; see if any char already struck
01FC CD0000C3F5                 call ?conin ! jmp u$conin$echo  ; yes, eat it and try again
                        u$cl:
0202 CD0000F5                   call ?conin ! push psw
0206 4FCD0000                   mov c,a ! call ?cono
020A F1FE61D8                   pop psw ! cpi 'a' ! rc
020E D620                       sui 'a'-'A'                 ; make upper case
0210 C9                         ret

0211            disk$command    ds      1       ; current wd1797 command
0212            select$mask     ds      1       ; current drive select code
0213            old$select      ds      1       ; last drive selected
0214            old$track       ds      1       ; last track seeked to

0215            disk$status     ds      1       ; last error status code for messages

0216 1020       select$table    db      0001$0000b,0010$0000b ; for now use drives C and D

                                ; error message components

0218 2C20526561read$msg         db      ', Read',0
021F 2C20577269write$msg        db      ', Write',0

0227 1802       operation$name  dw      read$msg

                                ; table of pointers to error message strings
                                ;       first entry is for bit 7 of 1797 status byte

0229 3902       error$table     dw      b7$msg
022B 4502                       dw      b6$msg
022D 4F02                       dw      b5$msg
022F 5702                       dw      b4$msg
0231 6A02                       dw      b3$msg
0233 7002                       dw      b2$msg
0235 7C02                       dw      b1$msg
0237 8302                       dw      b0$msg

0239 204E6F7420b7$msg           db      ' Not ready,',0
0245 2050726F74b6$msg           db      ' Protect,',0
024F 204661756Cb5$msg           db      ' Fault,',0
0257 205265636Fb4$msg           db      ' Record not found,',0
026A 204352432Cb3$msg           db      ' CRC,',0
0270 204C6F7374b2$msg           db      ' Lost data,',0
027C 2044524551b1$msg           db      ' DREQ,',0
0283 2042757379b0$msg           db      ' Busy,',0

028A 2052657472error$msg        db      ' Retry (Y/N) ? ',0

                                ; command string for Z80DMA device for normal operation

029A C3         dma$block       db      0C3h    ; reset DMA channel
029B 14                         db      14h     ; channel A is incrementing memory
029C 28                         db      28h     ; channel B is fixed port address
029D 8A                         db      8Ah     ; RDY is high, CE/ only, stop on EOB
029E 79                         db      79h     ; program all of ch. A, xfer B->A (temp)
029F            zdma$dma        ds      2       ; starting DMA address
02A1 7F00                       dw      128-1   ; 128 byte sectors in SD
02A3 85                         db      85h     ; xfer byte at a time, ch B is 8 bit address
02A4 07                         db      p$fddata ; ch B port address (1797 data port)
02A5 CF                         db      0CFh    ; load B as source register
02A6 05                         db      05h     ; xfer A->B
02A7 CF                         db      0CFh    ; load A as source register
02A8            zdma$direction  ds      1       ; either A->B or B->A
02A9 CF                         db      0CFh    ; load final source register
02AA 87                         db      87h     ; enable DMA channel
0011 =          dmab$length     equ     $-dma$block

02AB C3         read$id$block   db      0C3h    ; reset DMA channel
02AC 14                         db      14h     ; channel A is incrementing memory
02AD 28                         db      28h     ; channel B is fixed port address
02AE 8A                         db      8Ah     ; RDY is high, CE/ only, stop on EOB
02AF 7D                         db      7Dh     ; program all of ch. A, xfer A->B (temp)
02B0 1100                       dw      id$buffer ; starting DMA address
02B2 0500                       dw      6-1     ; Read ID always xfers 6 bytes
```

**Listing I-4. (continued)**

```
  424   02B4 85                             db      85h      ; byte xfer, ch B is 8 bit address
  425   02B5 07                             db      p$fddata ; ch B port address (1797 data port)
  426   02B6 CF                             db      0CFh     ; load dest (currently source) register
  427   02B7 01                             db      01h      ; xfer B->A
  428   02B8 CF                             db      0CFh     ; load source register
  429   02B9 87                             db      87h      ; enable DMA channel
  430   000F =          length$id$dmab  equ     $-read$id$block
  431
  432                            cseg    ; easier to put ID buffer in common
  433
  434   0011            id$buffer        ds      6        ; buffer to hold ID field
  435                            ; track
  436                            ; side
  437                            ; sector
  438                            ; length
  439                            ; CRC 1
  440                            ; CRC 2
  441
  442   0017                     end
B0MSG           0283    381    390#
B1MSG           027C    380    389#
B2MSG           0270    379    388#
B3MSG           026A    378    387#
B4MSG           0257    377    386#
B5MSG           024F    376    385#
B6MSG           0245    375    384#
B7MSG           0239    374    383#
BC              0000
BELL            0007     52#
CANCEL          01A6    289#
CHECKSEEK       01A9    226    257    296#
CR              000D     50#
DE              0002
DISKCOMMAND     0211    203    249    354#
DISKSTATUS      0215    251    275    359#
DMABLENGTH      0011    239    413#
DMABLOCK        029A    238    398#   413
DPBSD           0000     62     66     79     83     93#
ERRM1           0186    277#   281
ERRORMSG        028A    283    392#
ERRORTABLE      0229    276    374#
EXECCOMMAND     01D5    250    310    316    323#   337
FDINIT0         00BE     60    143#
FDINIT1         00CD     77    152#
FDINITNEXT      00C1    145#   150
FDLOGIN         00DB     59     76    162#
FDREAD          00DC     58     75    188#
FDSD0           000A     14     62#
FDSD1           005C     14     79#
FDWRITE         00E6     57     74    193#
HARDERROR       01A3    263    286#
HL              0004
IDBUFFER        0011    339    422    434#
IDOK            01BE    299    302    305#
INITTABLE       00CE    144    155#
IX              0004
IY              0004
LENGTHIDDMAB    000F    333    430#
LF              000A     51#
MORERETRIES     010D    210#   285
NEWTRACK        012D    217    221    225#
OLDSELECT       0213    215    356#
OLDTRACK        0214    219    357#
OPERATIONNAME   0227    202    271    369#
PBANKSELECT     0025    243    247
PBAUDCON1       000C
```

**Listing I-4. (continued)**

```
PBAUDCON2       0030
PBAUDCON34      0031
PBAUDLPT1       000E
PBAUDLPT2       0032
PBOOT           0014
PCENTDATA       0011
PCENTSTAT       0010
PCON2DATA       002C
PCON2STAT       002D
PCON3DATA       002E
PCON3STAT       002F
PCON4DATA       002A
PCON4STAT       002B
PCONFIGURATION  0024
PCRTDATA        001C
PCRTSTAT        001D
PFDCMND         0004   325
PFDDATA         0007   308    406    425
PFDINT          0008   327
PFDMISC         0009   223
PFDSECTOR       0006   236
PFDSTAT         0004   328
PFDTRACK        0005   235    306
PINDEX          000F
PLPT2DATA       0028
PLPT2STAT       0029
PLPTDATA        001E
PLPTSTAT        001F
PRTC            0033
PSELECT         0008   209
PWD1797         0004
PZCTC1          000C
PZCTC2          0030
PZDART          001C
PZDMA           0000   239    333
PZPIO1          0008
PZPIO1A         000A   155
PZPIO1B         000B   157
PZPIO2          0010
PZPIO2A         0012
PZPIO2B         0013
PZPIO3          0024
PZPIO3A         0026
PZPIO3B         0027
PZSIO1          0028
PZSIO2          002C
READID          01E1   298    301    331#
READIDBLOCK     02AB   332    417#   430
READMSG         0218   189    366#   369
RESTORE         01D3   303    318#
RETRYOPERATION  010F   212#   259
RWCOMMON        00ED   191    198#
SAMETRACK       0139   223    234#
SELECTMASK      0212   208    215    355#
SELECTTABLE     0216   207    361#
SPINLOOP        0133   229#   232
STEPOUT         01CE   300    314#
TRANS           00A4    62     63     79     80    106#
UC1             0202   344    346#
UCONINECHO      01F5   284    343#   345
WAITIREQ        01D7   326#   327
WRITEMSG        021F   194    367#
ZDMADIRECTION   02A8   204    410#
ZDMADMA         029F   205    403#
?CONIN          0000    32    345    347
?CONO           0000    32    348
?CONST          0000    33    344
?PDEC           0000    30
?PDERR          0000    31    269
?PMSG           0000    29    271    280    283
?WBOOT          0000    28    290
@ADRV           0000    18
```

**Listing I-4. (continued)**

```
@DBNK           0000    20   245
@DMA            0000    19   205
@ERMDE          0000    24   263
@RDRV           0000    18   206
@SECT           0000    19   236
@TRK            0000    19   219   235   307
```

**Listing I-4. (continued)**

## I.5 Bank and Move Module for CP/M 3 Linked BIOS

The MOVE.ASM module performs memory-to-memory moves and bank selects.

```
                                title 'bank & move module for CP/M3 linked BIOS'

                                cseg

                                public ?move,?xmove,?bank
                                extrn @cbnk

                                maclib z80
                                maclib ports

                        ?xmove:         ; ALTOS can't perform interbank moves
     0000 C9                    ret

                        ?move:
     0001 EB                    xchg            ; we are passed source in DE and dest in HL
                                ldir            ; use Z80 block move instruction
     0002+EDB0                  DB      0EDH,0B0H
     0004 EB                    xchg            ; need next addresses in same regs
     0005 C9                    ret

                                                                ; by exiting through bank select
                        ?bank:
     0006 C5                    push b                          ; save register b for temp
     0007 171717E618            ral ! ral ! ral ! ani 18h       ; isolate bank in proper bit position
     000C 47                    mov b,a                         ; save in reg B
     000D DB25                  in p$bankselect                 ; get old memory control byte
     000F E6E7B0                ani 0E7h ! ora b                ; mask out old and merge in new
     0012 D325                  out p$bankselect                ; put new memory control byte
     0014 C1                    pop b                           ; restore register b
     0015 C9                    ret

                                                        ;       128 bytes at a time

     0016                       end
BC              0000
DE              0002
HL              0004
IX              0004
IY              0004
PBANKSELECT     0025    26    28
PBAUDCON1       000C
PBAUDCON2       0030
PBAUDCON34      0031
PBAUDLPT1       000E
PBAUDLPT2       0032
PBOOT           0014
PCENTDATA       0011
PCENTSTAT       0010
PCON2DATA       002C
PCON2STAT       002D
PCON3DATA       002E
PCON3STAT       002F
PCON4DATA       002A
PCON4STAT       002B
PCONFIGURATION  0024
PCRTDATA        001C
```

**Listing I-5. Bank and Move Module for CP/M 3 Linked BIOS**

```
PCRTSTAT        001D
PFDCMND         0004
PFDDATA         0007
PFDINT          0008
PFDMISC         0009
PFDSECTOR       0006
PFDSTAT         0004
PFDTRACK        0005
PINDEX          000F
PLPT2DATA       0028
PLPT2STAT       0029
PLPTDATA        001E
PLPTSTAT        001F
PRTC            0033
PSELECT         0008
PWD1797         0004
PZCTC1          000C
PZCTC2          0030
PZDART          001C
PZDMA           0000
PZPIO1          0008
PZPIO1A         000A
PZPIO1B         000B
PZPIO2          0010
PZPIO2A         0012
PZPIO2B         0013
PZPIO3          0024
PZPIO3A         0026
PZPIO3B         0027
PZSIO1          0028
PZSIO2          002C
?BANK           0006      5     22#
?MOVE           0001      5     14#
?XMOVE          0000      5     11#
@CBNK           0000      6
```

**Listing I-5. (continued)**

## I.6 I/O Port Addresses for Z80 Chip-based System: PORTS.LIB

This listing is the PORTS.LIB file on your distribution diskette. It contains the port addresses for the Z80 chip-based system with a Western Digital 1797 Floppy Disk Controller.

```
        I/O Port addresses for Z80 chip set based system with wd1797 FDC

                ; chip bases

p$zdma          equ 0
p$wd1797        equ 4
p$zpio1         equ 8
p$zctc1         equ 12
p$zpio2         equ 16
p$boot          equ 20   ; OUT disables boot EPROM
p$zdart         equ 28   ; console 1 and printer 1
p$zpio3         equ 36
p$zsio1         equ 40
p$zsio2         equ 44
p$zctc2         equ 48

                ; diskette controller chip ports

p$fdcmnd        equ p$wd1797+0
p$fdstat        equ p$wd1797+0
p$fdtrack       equ p$wd1797+1
p$fdsector      equ p$wd1797+2
p$fddata        equ p$wd1797+3

                ; parallel I/O 1
```

**Listing I-6. I/O Port Addresses for Z80 Chip-based System**

```
p$select        equ p$zpio1+0
p$fdint         equ p$zpio1+0
p$fdmisc        equ p$zpio1+1
p$zpio1a        equ p$zpio1+2
p$zpio1b        equ p$zpio1+3
        ; counter timer chip 1

p$baudcon1      equ p$zctc1+0
p$baudlpt1      equ p$zctc1+2
p$index         equ p$zctc1+3


        ; parallel I/O 2, Centronics printer interface

p$cent$stat     equ p$zpio2+0
p$cent$data     equ p$zpio2+1
p$zpio2a        equ p$zpio2+2
p$zpio2b        equ p$zpio2+3


        ; dual asynch rcvr/xmtr, console and serial printer ports
p$crt$data      equ p$zdart+0
p$crt$stat      equ p$zdart+1
p$lpt$data      equ p$zdart+2
p$lpt$stat      equ p$zdart+3


        ; Third Parallel I/O device

p$configuration equ p$zpio3+0
p$bankselect    equ p$zpio3+1
p$zpio3a        equ p$zpio3+2
p$zpio3b        equ p$zpio3+3


        ; Serial I/O device 1, printer 2 and console 4

p$lpt2data      equ p$zsio1+0
p$lpt2stat      equ p$zsio1+1
p$con4data      equ p$zsio1+2
p$con4stat      equ p$zsio1+3


        ; Serial I/O device 2, console 2 and 3

p$con2data      equ p$zsio2+0
p$con2stat      equ p$zsio2+1
p$con3data      equ p$zsio2+2
p$con3stat      equ p$zsio2+3


        ; second Counter Timer Circuit

p$baudcon2      equ p$zctc2+0
p$baudcon34     equ p$zctc2+1
p$baudlpt2      equ p$zctc2+2
p$rtc           equ p$zctc2+3
```

**Listing I-6. (continued)**

## I.7 Sample Submit File for ASC 8000-15 System

Digital Research used this SUBMIT file to build the sample BIOS.

```
;Submit file to build sample BIOS for ACS 8000-15 single-density system
;
rmac bioskrnl
rmac boot
rmac move
rmac chario
rmac drvtbl
rmac fd1797sd
rmac scb
link bnkbios3[b,q]=bioskrnl,boot,move,chario,drvtbl,fd1797sd,scb
gencpm
```

**Listing I-7. Sample Submit File for ASC 8000-15 System**

End of Appendix I

# Appendix J
# Public Entry Points for CP/M 3 Sample BIOS Modules

| Module Name | Public Entry Point | Function | Input Parameter | Return Value |
|---|---|---|---|---|
| BIOSKRNL | | | | |
| | ?PMSG | Print Message | HL points to msg | none |
| | ?PDEC | Print Decimal | HL=number | none |
| | ?PDERR | Print BIOS Disk Err Msg Header | none | none |
| CHARIO | | | | |
| | ?CINIT | Char Dev Init | C=Phys Dev # Dev Parms in @CTBL | none |
| | ?CIST | Char Inp Dev St | B=Phys Dev # | A=00 if no input A=0FFH if input char available |
| | ?COST | Char Out Dev St | B=Phys Dev # | A=00 if output busy A=0FFH if output ready |
| | ?CI | Char Dev Input | B=Phys Dev # | A=next available input char |
| | ?CO | Char Dev Output | B=Phys Dev # C=Input Char | |
| MOVE | | | | |
| | ?MOVE | Memory to Memory Move | BC=byte count DE=start source adr HL=start dest adr | DE,HL point to next bytes after move |
| | ?XMOVE | Set Banks for Extended Move | B=Dest Bank C=Source Bank | BC,DE,HL are unchanged |
| | ?BANK | Select Bank | A=Bank Number | All unchanged |
| BOOT | | | | |
| | ?INIT | System Init | none | none |
| | ?LDCCP | Load CCP | none | none |
| | ?RLCCP | Reload CCP | none | none |
| | ?TIME | Get/Set Time | C=000H if get C=0FFH if set | none |

**Listing J-1. Public Entry Points for CP/M 3 Sample BIOS Modules**

End of Appendix J

# Appendix K
# Public Data Items in CP/M 3 Sample BIOS Modules

**Table K-1. Public Data Items**

| Module Name | Public Data | Description |
|---|---|---|
| BIOSKRNL | | |
| | @ADRV | Absolute Logical Drive Code |
| | @RDRV | Relative logical drive code (UNIT) |
| | @TRK | Track Number |
| | @SECT | Sector Address |
| | @DMA | DMA Address |
| | @DBNK | Bank for Disk I/O |
| | @CNT | Multi-sector Count |
| | @CBNK | Current CPU Bank |
| CHARIO | | |
| | @CTBL | Character Device Table |
| DRVTBL | | |
| | @DTBL | Drive Table |

End of Appendix K

# Appendix L
# CP/M 3 BIOS Function Summary

**Table L-1. BIOS Function Jump Table Summary**

| No. | Function | Input | Output |
|---|---|---|---|
| 0 | BOOT | None | None |
| 1 | WBOOT | None | None |
| 2 | CONST | None | A=0FFH if ready<br>A=00H if not ready |
| 3 | CONIN | None | A=Con Char |
| 4 | CONOUT | C=Con Char | None |
| 5 | LIST | C=Char | None |
| 6 | AUXOUT | C=Char | None |
| 7 | AUXIN | None | A=Char |
| 8 | HOME | None | None |
| 9 | SELDSK | C=Drive 0-15<br>E=Init Sel Flag | HL=DPH addr<br>HL=000H if invalid dr. |
| 10 | SETTRK | BC=Track No | None |
| 11 | SETSEC | BC=Sector No | None |
| 12 | SETDMA | BC=.DMA | None |
| 13 | READ | None | A=00H if no Err<br>A=01H if Non-recov Err<br>A=0FFH if media changed |
| 14 | WRITE | C=Deblk Codes | A=00H if no Err<br>A=01H if Phys Err<br>A=02H if Dsk is R/O<br>A=0FFH if media changed |
| 15 | LISTST | None | A=00H if not ready<br>A=0FFH if ready |
| 16 | SECTRN | BC=Log Sect No<br>DE=Trans Tbl Adr | HL=Phys Sect No |
| 17 | CONOST | None | A=00H if not ready<br>A=0FFH if ready |
| 18 | AUXIST | None | A=00H if not ready<br>A=0FFH if ready |
| 19 | AUXOST | None | A=00H if not ready<br>A=0FFH if ready |
| 20 | DEVTBL | None | HL=Chrtbl addr |
| 21 | DEVINI | C=Dev No 0-15 | None |
| 22 | DRVTBL | None | HL=Drv Tbl addr<br>HL=0FFFFH<br>HL=0FFFEH |
| 23 | MULTIO | C=Mult Sec Cnt | None |
| 24 | FLUSH | None | A=000H if no err<br>A=001H if phys err<br>A=002H if disk R/O |
| 25 | MOVE | HL=Dest Adr<br>DE=Source Adr<br>BC=Count | HL & DE point to next bytes following MOVE |

**Table L-1. (continued)**

| No. | Function | Input | Output |
|---|---|---|---|
| 26 | TIME | C=Get/Set Flag | None |
| 27 | SELMEM | A=Mem Bank | None |
| 28 | SETBNK | A=Mem Bank | None |
| 29 | XMOVE | B=Dest Bank<br>C=Source Bank | None |
| 30 | USERF | Reserved for System Implementor | |
| 31 | RESERV1 | Reserved for Future Use | |
| 32 | RESERV2 | Reserved for Future Use | |

End of Appendix L

# Index

**D**

## S



## T



## U



## V

# NOTES

# NOTES

# NOTES

# NOTES

# NOTES

# NOTES

## NOTES

# Reader Comment Card

We welcome your comments and suggestions. They help us provide you with better product documentation.

1. What sections of this manual are especially helpful?

2. What suggestions do you have for improving this manual? What information is missing or incomplete? Where are examples needed?

3. Did you find errors in this manual? (Specify section and page number.)

**CP/M Plus™ (CP/M® Version 3) Operating System User's Guide**
**Second Edition: March 1983**
**1006-2003-002**

From: ______________________

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO. 251 HOLMES, PA

POSTAGE WILL BE PAID BY ADDRESSEE

**Commodore Business Machines**
**CP/M Comments**
Box 651
Holmes, PA 19043

# Reader Comment Card

We welcome your comments and suggestions. They help us provide you with better product documentation.

1. What sections of this manual are especially helpful?

2. What suggestions do you have for improving this manual? What information is missing or incomplete? Where are examples needed?

3. Did you find errors in this manual? (Specify section and page number.)

CP/M Plus™ (CP/M® Version 3) Operating System Programmer's Guide
Second Edition: April 1983
1006-2023-002

From: ____________________

____________________

____________________

____________________

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO. 251 HOLMES, PA

POSTAGE WILL BE PAID BY ADDRESSEE

**Commodore Business Machines**
**CP/M Comments**
Box 651
Holmes, PA 19043

# Reader Comment Card

We welcome your comments and suggestions. They help us provide you with better product documentation.

1. What sections of this manual are especially helpful?

2. What suggestions do you have for improving this manual? What information is missing or incomplete? Where are examples needed?

3. Did you find errors in this manual? (Specify section and page number.)

**CP/M Plus™ (CP/M® Version 3) Operating System System Guide**
**Second Edition: July 1983**
**1006-2013-002**

From: ____________________

____________________

____________________

____________________

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO. 251 HOLMES, PA

POSTAGE WILL BE PAID BY ADDRESSEE

**Commodore Business Machines**
**CP/M Comments**
Box 651
Holmes, PA 19043

319791-01